CHORALEARN: A SYSTEM THAT LEARNS FROM EXAMPLES TO WRITE SIMPLIFIED FIGURED BASS HARMONIZATIONS OF CHORALES USING THE PLS1 CLUSTERER

BY

DAVID WINSOR SIRKIN

B.A., Amherst College, 1976
Ph.D., University of Illinois, 1982

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

THE GRADUATE COLLEGE

NOVEMBER 1986

WE HEREBY RECOMMEND THAT THE THESIS BY

DAVID WINSOR SIRKIN

ENTITLED CHORALEARN: A SYSTEM THAT LEARNS FROM EXAMPLES TO WRITE SIMPLIFIED FIGURED BASS HARMONIZATIONS OF CHORALES USING THE PLS1 CLUSTERER

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

Director of Thesis Research

Edward Mast

for Head of Department

Committee on Final Examination†

Chairperson

† Required for doctor's degree but not for master's.

O-517

ACKNOWLEDGEMENTS

First of all, I thank my advisor, Professor Larry Rendell. The PLS1 clusterer is his creation. He also made very useful criticisms of the manuscript, and helped me develop the method for handling feedback from a Bach expert, which is proposed in Section V. Christopher Matheus also contributed a valuable suggestion for that method.

I thank David Tcheng for writing the program that allowed the computer to really "listen" to Bach being played (very slowly) on his synthesizer.

I thank my other fellow workers in the Artificial Intelligence Laboratory for helping me with Unix and otherwise answering the questions of one new to the programming business. A special thanks goes to Raymond Board, who, in addition to helping me with some programming problems, also played some of the computer's first harmonizations on the piano and recorded them on tape for me. A special thanks also to Raj Seshu, whose encouragement may have been crucial for the metamorphosis of my original idea into a successful project.

I thank the computer operating staff of the Department of Computer Science, especially Mike Schwager and George Karahalios for their assistance with all kinds of weird problems — from taping to dealing with sudden zaps. . . .

I thank Lippold Haken, Kurt Hebel, Carla Scaletti and Edward Carney of CERL Music/PLATO for preparing the music notation graphics in this thesis with their Interactive Music System and for providing a forum for the first public demonstration of ChoraLearn. I also thank Diane Cook for some other music graphics made with her system at the Computer Music Laboratory of the School of Music.

I thank Professor Jerald Hamilton for playing some of ChoraLearn's simplified figured bass harmonizations on the organ at the first public demonstration of the system, and also

for valuable discussions. For the latter I thank also Professors James Beauchamp, Thomas Fredrickson, Salvatore Martirano, and Robert Stepp.

I thank Lilian Beck for her careful reading of the manuscript, and finally, I thank Sharon Collins for typing and for putting up with all my nonsense.

PREFACE

This thesis presents a software system, ChoraLearn, that applies a general inductive learning program, the PLS1 clusterer, to the problem of harmonization in music. The work described represents the first time that a harmonizing program has induced rules from examples. Thus far the examples have been from the chorale harmonizations of J.S. Bach.

I have tried to accommodate more than one kind of reader, recognizing that there will only be a few who have a strong background (or a strong interest) in both machine learning and music theory. This preface gives some indication (and there are additional guidelines in the body of the thesis) as to what parts of the thesis are aimed at all readers, and what parts are likely to interest only some. The latter parts can be skipped without loss of continuity.

Section I of the thesis is a brief introduction that first summarizes previous work on harmonization by computer, and then explains what the PLS1 clusterer is.

Section II defines some terms, and then outlines what is involved in harmonizing a chorale melody. It goes on to specify the particular piece of the task that the current system has been learning, which can be described in musical terms as recognizing good two-chord progressions.

Section III explains how the system works. Part A explains how the system learns, i.e., how the Bach training examples are prepared for the clusterer, and what the clusterer does with them. It goes on to explain how the learned rules, as represented in the output of the clusterer, are utilized by the system in harmonizing a melody. Part B lists the features that are used by the system to describe two-chord progressions. A good choice of features is the key to successful learning with the clusterer, because the clusterer deals exclusively with descriptions that are in terms of the chosen features. However, the reader with no background in harmony may wish to skip this part, and still will be able to follow the rest of

the thesis.

Section IV presents some harmonizations by ChoraLearn and attempts to evaluate the progress the system has made so far in learning to recognize Bach-like two-chord progressions. Part E discusses issues that may interest some readers more than others, and may be skipped without loss of continuity.

Section V is a proposal for a way to supplement the current learning-from-examples machinery with a method for learning from a teacher who can recognize errors in the machine's performance. The proposed methodology may make inductive learning with the PLS1 clusterer more practical in other problem domains too.

Section VI relates to Section II in considering the task learned by the present system in the context of the complete task of chorale-style harmonization.

TABLE OF CONTENTS

I. INTRODUCTION

The first two sections of this introduction summarize previous work relating to harmonization by computer. The third section explains briefly what the PLS1 clusterer is. In the fourth and final section I explain what our objectives were in developing the ChoraLearn harmonization system.

A. Computers and Music

Computers have been used in various areas within the broad field of music, including analysis, instruction, composition, and electronic sound production. (Hiller, 1970; Roads, 1980; Gross 1984). Computer composition began here at the University of Illinois with the composition of the Illiac Suite in the late 1950's (Hiller and Isaacson, 1959; Hiller, 1981). Artificial intelligence techniques of induction and pattern recognition have been applied to some problems in music (Roads, 1980; Michalski and Stepp, 1983), but apparently not to composition or harmonization.

B. Computer Harmonization

1. The task

Harmonization of a chorale or hymn tune is an exercise confronted by nearly every student beginning the study of music theory. The task is to take a given melody (soprano line) and to write 3 parts of music lying below it in pitch for the alto, tenor, and bass, such that the four voices together produce a sequence of chords that is pleasing to the ear. Typically, there is a new chord on every quarter-note beat.[1] Chorale harmonizations by Bach, of which more than 400 have been collected (Terry, 1929), are often given to the

---

[1]At any given moment, however, one or more of the four lines of music may be moving (changing notes) at a faster than quarter-note rhythm. Then the chord for the beat may exist only for the first or second half of the beat, the other half containing notes not in the chord, or "nonharmonic tones."

student as good examples. The student, even when strictly adhering to the rules and guidelines found in standard texts such as Piston's (1962), usually produces much poorer results.

2. Previously developed systems

Because the textbook rules by themselves are inadequate to achieve Bach-like chorale harmonizations, the task of harmonizing a chorale tune seems to be one that would be appropriate for a computer system that performs inductive inference, or learns from examples. However, previous attempts to produce good (in one case specifically Bach-like) chorale harmonizations by computer have used exclusively implementations of rules found in books and additional rules provided by the programmers (rather than machine-induced rules). The most recent of these are the systems of Ebcioglu (1984, 1986) and Thomas (1985). Besides these there was a system of D. G. Champernowne created around 1960 (described in Hiller, 1970). A system that performed a task similar to harmonization of a melody, namely turning an unfigured bass into a figured bass, was developed by Rothgeb in the 1960's (Rothgeb, 1980).

The previous system closest to ours in conception was written by another University of Illinois student, Alberto Segre, while he was on a Fulbright fellowship in Milan, Italy, in 1980 (Segre, 1981). His system also examined examples of Bach chorales. Although it was designed to generate chorales, including the melodies, Segre could have modified it to make it able to harmonize a given melody, provided the melody were similar enough to the Bach examples (i.e., the melody would have to move by intervals that occurred in the examples). However, Segre's system was a rote learning system, meaning that it was restricted to rearranging elements that occurred in the Bach examples. Also, it was designed to handle

only a small number of Bach examples,[2] thus limiting the range of its possible outputs. Our system, because it uses the PLS1 clusterer, a modern machine learning tool, can accommodate a much larger number of examples, and can make generalizations; it is not restricted to using elements found in the examples. The latter ability is obviously an advantage for any music composition system, provided of course that the new elements resulting from the generalizations are consistent with the style of the elements in the training examples. This is the case with most of the new elements generated by the current ChoraLearn system.

C. The PLS1 Clusterer

PLS1 is an implementation of a PLS (probabilisic learning system) described in Rendell (1983; 1986c). The heart of PLS1 is the clusterer. Its purpose is to create a classification scheme for a particular category of *objects*. For example, the category of objects might be "plans for constructing aircraft." The clusterer might be given a collection of plans for aircraft, and would be told that each of them, if implemented, would result in a viable product (an aircraft that flies). It would also be given another collection that it would understand to be for aircraft that would never get off the ground. Its job would be to create a classification scheme for the objects (aircraft plans), such that when it is presented with a new object it can classify it, and then conclude that because it falls into class X it has probability P_X of being a plan for a viable aircraft. Creating the classification scheme is called *learning from examples.* When a system uses such a classification scheme in performing

---

[2]Segre ran his system with 4 Bach chorales as examples (Segre, personal communication). His system generated all possible chorales that could be constructed with elements from those examples before choosing one of them. Therefore, increasing the number of examples, and thereby increasing the number of available elements, would have caused a combinatorial explosion that would have drastically increased the time to generate chorales. ChoraLearn, on the other hand, does not take a significantly longer time to harmonize a phrase if its set of training examples is enlarged. This is because the "GoodChord" choosing procedure enables the system to generate a small number of harmonizations, with a high probability that some of them will be good (see Sections IV.B and IV.G). Also, increasing the number of training examples does not necessarily increase the number of choices for a chord (see Section IV.D).

some task, we speak of *performing*.

The clusterer does not actually deal with objects themselves, but rather with descriptions of them. The descriptions must be in terms of a predetermined, finite set of *features,* and each feature must have a finite set of *values*. For example, a feature might be "ratio of wingspan to fuselage length," and the possible values might be 1, 2, and 3, representing "less than one," "one," and "greater than one," respectively. An object description can thus be thought of as a *feature vector* having dimension equal to the number of predetermined features. From this point of view, each feature is a dimension of a *feature space*. The two collections of objects ("good" and "bad"[3]) are then two sets of vectors, or points, in a feature space. What the clusterer does is partition the feature space such that good objects with similar descriptions are grouped together and isolated as much as possible from bad objects. The result is that the feature space is divided into *regions,* each region having an associated *utility* (also called "probability," because it is simply the number of good points in the region divided by the number of total points in the region, and therefore is the probability that a point in the region is good).

Rendell (1986a) considers some ways that learning systems that use the above-described feature space methodology could be extended and made more powerful.

D. Objectives of This Project

In ChoraLearn, the system described in this thesis, we have used the PLS1 clusterer to induce rules of harmonization from examples from Bach (the rules are represented in a set of feature space regions that result from clustering). We will present evidence that the clusterer has achieved useful learning for the harmonizing system. From the musical perspective, the

[3]The "good" collection may in fact contain a small percentage of bad objects and vice versa. See also Section III.A.1.

goal of the project has been to demonstrate that, using a modern machine learning tool, we can develop a system that induces its own rules of harmonization from a set of examples. Such a system has several advantages over rule-based systems, one of them being that the style of music it writes is changeable, being determined by the set of examples it is given to examine (other advantages will be discussed later).

From an artificial intelligence perspective, the goal is to demonstrate that the PLS1 clusterer, which has already been applied successfully in several other domains, is a general learning tool whose applicability is broad enough that it also can succeed with harmonization. As already mentioned (Section B.1), harmonization is a task that cannot be mastered easily. Although our current system does not write complete chorale harmonizations, the part of the task that it is on the way to mastering is the one which perhaps more than any other distinguishes a great harmonization from a mediocre one (see below, Section II.B.2).

II. DEFINITIONS OF TERMS AND TASKS

A. Definitions of Terms

Technical terms pertaining to learning with the PLS1 clusterer ("object," "feature," etc.) have already been defined (Section I.C). However, there are also several domain-specific (i.e., musical) terms used in this thesis. The reader who has not studied harmony need not be concerned with all of them. However, it is necessary that I make clear to both the musically knowledgeable and the musically naive reader how I will be using certain of the musical terms.

A *chord* is a set of notes sounded together (simultaneously), creating a specific sonority. Since we are dealing here with four-voiced chorales, a chord may be thought of as four notes sung by the sopranos, altos, tenors, and basses respectively. In this thesis the term chord will usually mean a melody (soprano) note and a bass note with an associated "figure," or instruction that partially specifies the alto and tenor notes. This might be called more correctly, "a *figured bass* specification for a chord," but I will simply call it a chord here.

Two kinds of chords will be mentioned in this thesis. It is sufficient that the musically naive reader know that these are called *triads* and *seventh chords,* and that a seventh chord can be thought of as being a triad with an additional element, called a *seventh,* added on to it.

I will use the term *progression* to mean a two-chord progression, or sequence, i.e., two chords, one following immediately after the other. The first chord of a progression will be called *Chord,* and the second *NextChord.* A progression will sometimes be referred to as an object, because progressions are the objects (see Section I.C above) that so far have been examined by the learning machinery of ChoraLearn.

A chorale *phrase* is a sequence of typically 4 to 10 chords (usually each chord having the duration of a quarter note). The last two or three chords of a phrase constitute the *cadence,* a final or temporary stopping point. Since the present ChoraLearn system does not harmonize cadences, I will sometimes use the term "phrase" to mean exclusively the noncadential portion of a phrase.

B. Defining the Computer's Task--Music Perspective

The task of harmonization was briefly explained in Section I.B.1. I will explain here more precisely what we wanted the computer to do. I will present this "definition of the task" twice, first in a simplified way for the reader who is not concerned with the musical details and rationales, and then more fully. Finally, I will briefly discuss the significance of the chosen task definition.

1. Simple explanation

The computer's task was to make a classification scheme for progressions, using a set of progressions taken from Bach's chorale harmonizations as the collection of "good objects" (see Section I.C above). It was then to construct harmonizations for a given melody (except for the last two or three notes, which belong to the cadence[4]) by joining together progressions that were Bach-like according to this classification scheme (and that also satisfied the constraints imposed by the particular melody). It would be left to another system (a "second stage of learning"—not yet implemented) to choose the best harmonizations for a short melody (soprano line of a phrase) by using a classification scheme for phrases. In other

---

[4]The task for the current system did not include the generation of a cadence. Rather, the first chord of the cadence (that a human would have had to write for the given melody) was given to the computer as a starting point from which to join progressions extending backwards towards the beginning of the phrase (the way the program did this is explained in Section III.A.2 below).

words, the objects for this other system would be phrases rather than individual progressions).

2. Musically detailed explanation

Harmonizing a chorale phrase is a process that is usually done in well-defined stages. First, some basic decisions must be made. One is what kind of cadence (end of phrase) formula to use. Another is what key (or keys) to harmonize the phrase in. By this I do not mean register (as in "Give me `Melancholy Baby´ in the key of G-flat"), but rather I am referring to the fact that a short sequence of notes (some chorale phrases are only 4 or 5 notes long) can be understood in more than one key. For example, the sequence f g a g can be understood as being in the key of C major, or F major, or B-flat major, to name only 3 of the possible keys. It is also possible to change keys (modulate) in the middle of a phrase.

After these decisions are made, the next step typically is to write what is called a figured bass. This is a bass line, with figures (numbers) indicating the harmony (chord type) that is to be formed by the four voices on each beat. Sometimes movement of a voice (or voices) on the half-beat is also indicated by the figures. The final step is then to fill in the inner voices (tenor and alto).

For the present study I chose one part of the chorale phrase harmonization process for the computer to learn. That part is the construction of a simplified figured bass harmonization (defined below) for the noncadential portion of a phrase. I chose the noncadential portion because the cadence tends to fall into one of a fairly small number of well-defined categories. Thus the cadence should probably be treated separately from the rest of the phrase, and, as a separate problem, it would seem to be easier (and less interesting) than the problem of harmonizing the rest. Writing the bass line (with figures) for

the noncadential portion of a phrase can be regarded as the heart of chorale harmonizing. More than any other part it is the one that distinguishes a great harmonization from an ordinary one. As Riemenschneider (1941) has noted, Bach's harmonizations owe their power especially to their strong bass lines. In fact, there are chorale melodies for which Bach wrote only a figured bass, leaving the organist to improvise the exact placing of the inner voices (Terry, 1929; Riemenschneider, 1941).

Figure 1a shows an example of a chorale phrase fully harmonized by Bach. Figure 1b shows his figured bass harmonization for the same chorale (and corresponding exactly to the full harmonization). I decided to simplify the task for the computer still further, by defining a "simplified figured bass." This is made by reducing the bass line to quarter-note harmonic tones and removing figures that pertain to eighth-note movement. The simplified version of the figured bass in Fig. 1b is shown in Fig. 1c. To illustrate the meaning of a figured bass to the reader who has not studied music theory I have also provided Fig. 1d, which shows explicitly the chords implied by the simplified figured bass in Fig. 1c. In other words, Figs. 1c and 1d contain the same information. However, the chords in Fig. 1d have in all cases the alto and tenor parts as close to the soprano as possible. This is a simple, regular way to meet the specifications given by the figured bass, but it often results in bad voice-leading. A composer, or an organist experienced in playing from a figured bass score, would not fill in the inner voices in such a regular fashion.

The computer system is supposed to learn from a set of examples taken exclusively from the chorale harmonizations of Bach. Thus the task for the computer can be summarized as follows: given a chorale phrase melody, the key in which to harmonize it in, and the cadence already determined, write a simplified figured bass for the rest of the phrase. The task is illustrated graphically in Fig. 1e. The first part of the figured bass in Fig. 1c is missing in

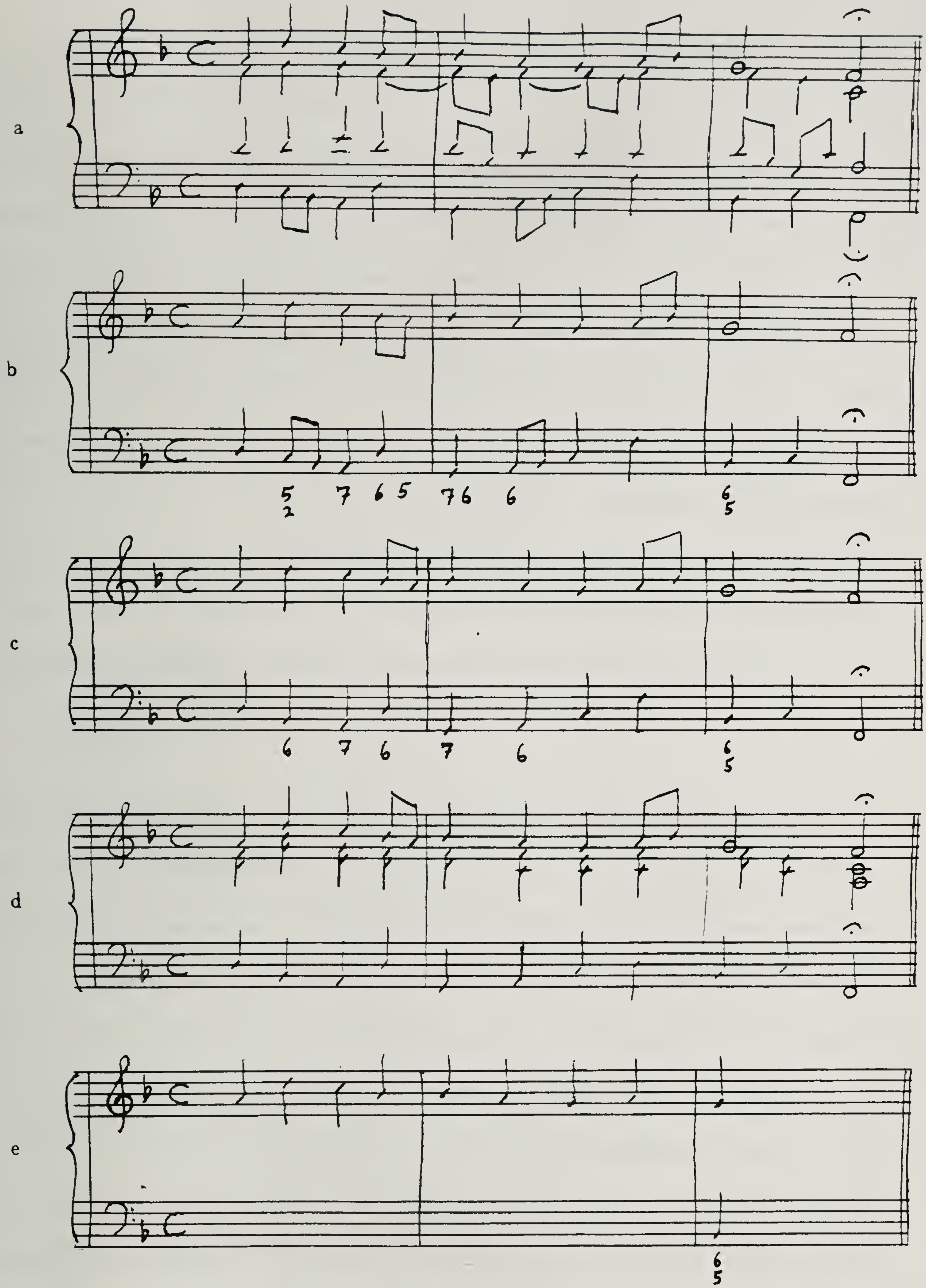


Figure 1. Several representations of a chorale phrase (a, b, c, d, and e are explained in the text).

Fig. 1e, and this is the job of the computer to write. The notes and figures in Fig. 1e are what is given to the computer as an input.

I restricted the problem still further by limiting the domain to phrases that are to be harmonized in a major key (the phrase in Fig. 1 is in minor) without modulation. Furthermore, I decided to divide the process of learning to write a good simplified figured bass into two stages. In the first stage the computer is to learn only what constitutes a good progression, i.e., going from one chord (as defined by the figured bass) to the next. A second stage of learning would deal with attributes of a whole phrase. For this study, only the first stage has been implemented. That is why the notes in Fig. 1e include only the first of the cadence chords. In a system that has completed only the first stage of learning, only the first cadence chord needs to be examined, and then only to write the figured bass for the soprano note immediately preceding it. The other cadence chords need never be examined.

3. Significance

The present study deals only with an isolated part of the chorale harmonization problem, but it is a fundamental part, which can be regarded as forming a foundation for much of the rest. It also seems to be one of the most difficult parts. It is an underlying premise of the present study that if the PLS1 clusterer can be used successfully to learn the restricted task I have defined, then it could also be used in a similar manner to learn most, if not all, of the other parts of the chorale harmonization problem, such that piecewise, the whole problem could be learned by machine. It would be unthinkable to use the PLS1 clusterer to learn all the parts at once. The overall size of the problem has been estimated by Ebcioglu (1986) to be almost 10^{300} (number of possible chorales for a typical chorale melody). As we will see, our current system uses the clusterer to choose from a set containing only about 40 objects (two-chord progressions). Yet to do this, it uses a feature space containing

about 10^7 points, which is large compared to most other feature spaces created for other applications of the clusterer.[5]

C. The Task as Concept Learning—Artificial Intelligence Perspective

1. Restating what is being learned

From the preceding section we can see that what the current ChoraLearn system does in terms of learning is learn the single concept, "Bach-like progression." What the system does with this learning is classify trial objects (two-chord progressions) as either Bach-like or non-Bach-like. The not yet implemented second stage of learning (discussed above, Section B) is to learn the single concept, "Bach-like phrase." The phrases it will be given to classify are those generated by the current system, phrases composed of progressions classified by the current system as Bach-like.

2. Assessing the amount of learning

After a learning system has been developed, it is the job of the researcher to determine whether the learning has been effective, and, if possible, to quantify the amount of learning. If the learning element of the system is the PLS1 clusterer, this job can be described also as, "determining the usefulness of the classification scheme arrived at by the clusterer" (see Section I.C).

As we will see below (Section III.A.2), the ChoraLearn system uses the classification scheme as a final step in limiting its choices of progressions. Here, I will just give a simplified summary of the procedure. The system uses "generate and test" to choose (two-chord) progressions. About 300 trial progressions are generated for each progression that the

---

[5]The size of our feature space is in fact strikingly large when considered together with the small number of objects that it is being used to choose from.

computer chooses to be part of a harmonization. The trial chords are tested, or screened, three times before those that remain are classified according to the scheme from the clusterer. This sequential reduction of the number of possible choices for a progression is represented graphically in Fig. 2. Referring to that figure, we see that measuring the usefulness of the classification scheme amounts to determining how much closer the set of progressions classified as Bach-like comes to being coincident with the set of truly Bach-like progressions than does the set of progressions submitted to be classified (reduced from the originally generated set by the preliminary screens). In other words, we want to determine how much the clusterer has accomplished towards learning the concept "Bach-like" progression. We have attempted to make this measurement in two ways: by recruiting a Bach expert to examine some of our system's output for quality, and by seeing if our system would be capable of generating some of the harmonizations Bach wrote. This evaluation of the learning done by the clusterer will be explained in Section IV.D.

It should be kept in mind that the current system strings together progressions to make phrases. The importance of what the current system has learned about progressions must ultimately be judged by how much that learning will contribute to the task of generating Bach-like phrases. An estimate of this contribution is part of the evaluation presented in Section IV.D. The number of possible ways to harmonize a phrase is obviously much greater than the number of ways to harmonize a single note (harmonizing a single note is what the current system does when it chooses a progression). This is discussed quantitatively in Sections IV.D. If phrases were to be constructed using progressions chosen randomly from one of the larger sets in Fig. 2, then obtaining a Bach-like phrase would be about as probable as obtaining Hamlet's soliloquy from the proverbial monkey at a typewriter. For example, with the original set of 300 progressions (largest set in Fig. 2), the number of possible phrases

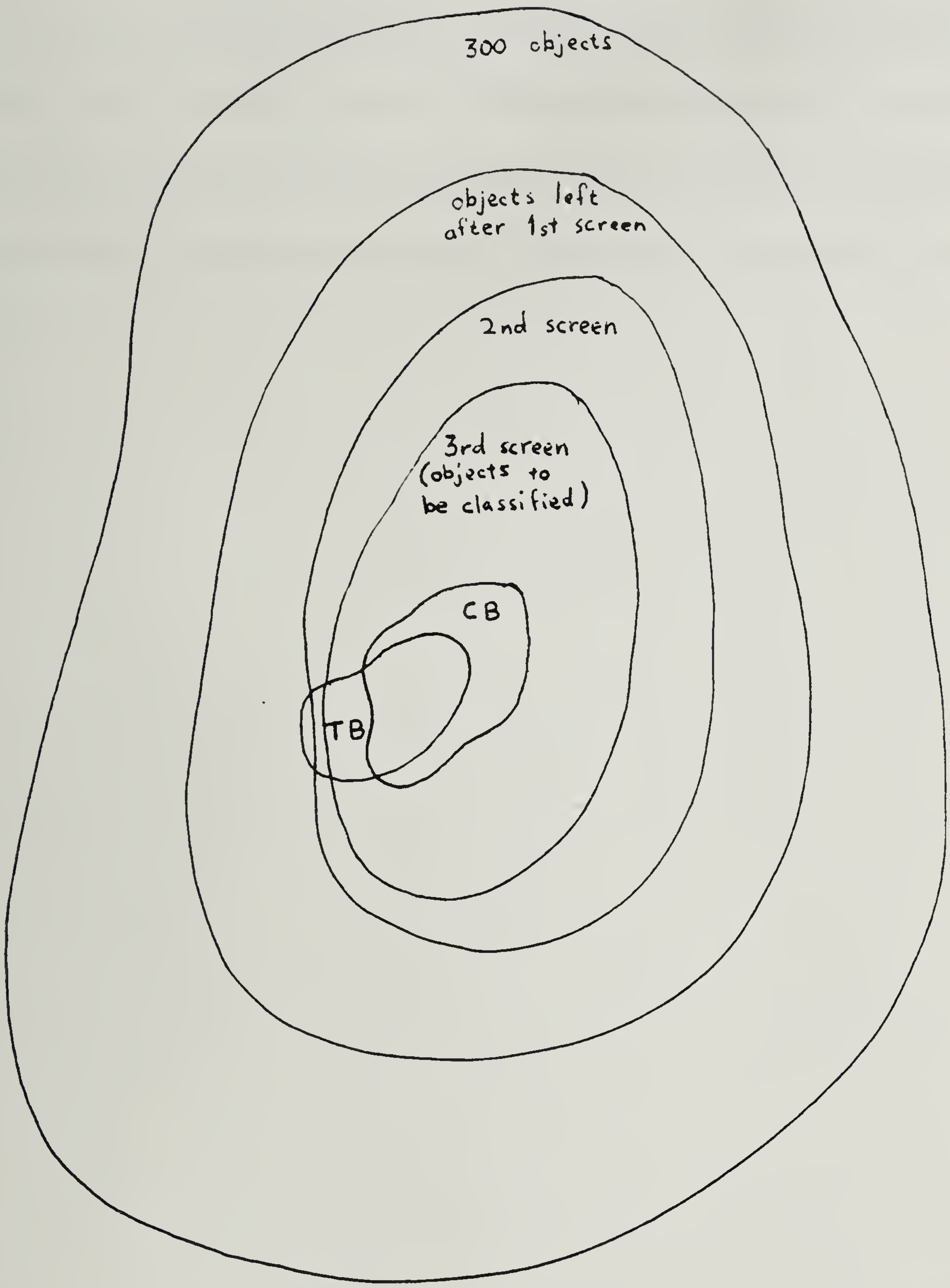


Figure 2. Graphical representation of the process of reducing the number of possible choices for a progression (object). The set marked CB is the set of objects classified as Bach-like according to the scheme created by the clusterer. The set marked TB is the set of objects that are truly Bach-like.

containing 5 (noncadential) progressions is 300^5, or 2×10^{12}. By limiting the choices for each progression to those that are for the most part Bach-like, the current system increases the likelihood of obtaining a Bach-like phrase. The achievement could be likened to giving the monkey a typewriter that typed a word out of Shakespeare's vocabulary for every key stroke. However, while the monkey's chances of replicating Hamlet's soliloquy would still be next to nil, our current system's chances of replicating a Bach phrase are not too bad, as we shall see in Section IV.C.

III. CHORALEARN: THE METHOD

A. Overview of the System

As explained by Buchanan et al. (1978), a learning system can be thought of as consisting of a few basic elements, two of which are a learning element and a performance element (see above, Section I.C). In ChoraLearn the learning so far has been entirely from Bach examples, and so there has been no interaction between the learning and performance elements except, obviously, that the performance element has utilized the output of the learning element. The overall system is schematized simply in Fig. 3. Figure 4 diagrams the system in more detail. The learning element consists of several Pascal programs: the PLS1 clusterer and programs to prepare the input files for the clusterer and to reformat the output for use by the performance element. The performance element is a single Pascal program that takes a melody and a first cadence chord (see Fig. 1e) and, with reference to the clusterer output, produces a simplified figured bass harmonization of the melody.

1. Learning

As explained in Section I.C, the clusterer needs to be given feature vectors for a collection of good objects and for a collection of bad objects in order to partition the feature space, and in so doing create a classification scheme. The good objects for our system were progressions from Bach chorale harmonizations. The collection of bad objects was created by having the computer generate all alternative progressions (within certain constraints) for each Bach progression. This collection included a small number of good progressions also, since, it is safe to say, there is usually more than one Bach-like progression that can be used in a particular place. Therefore we will no longer refer to the objects in this collection as bad

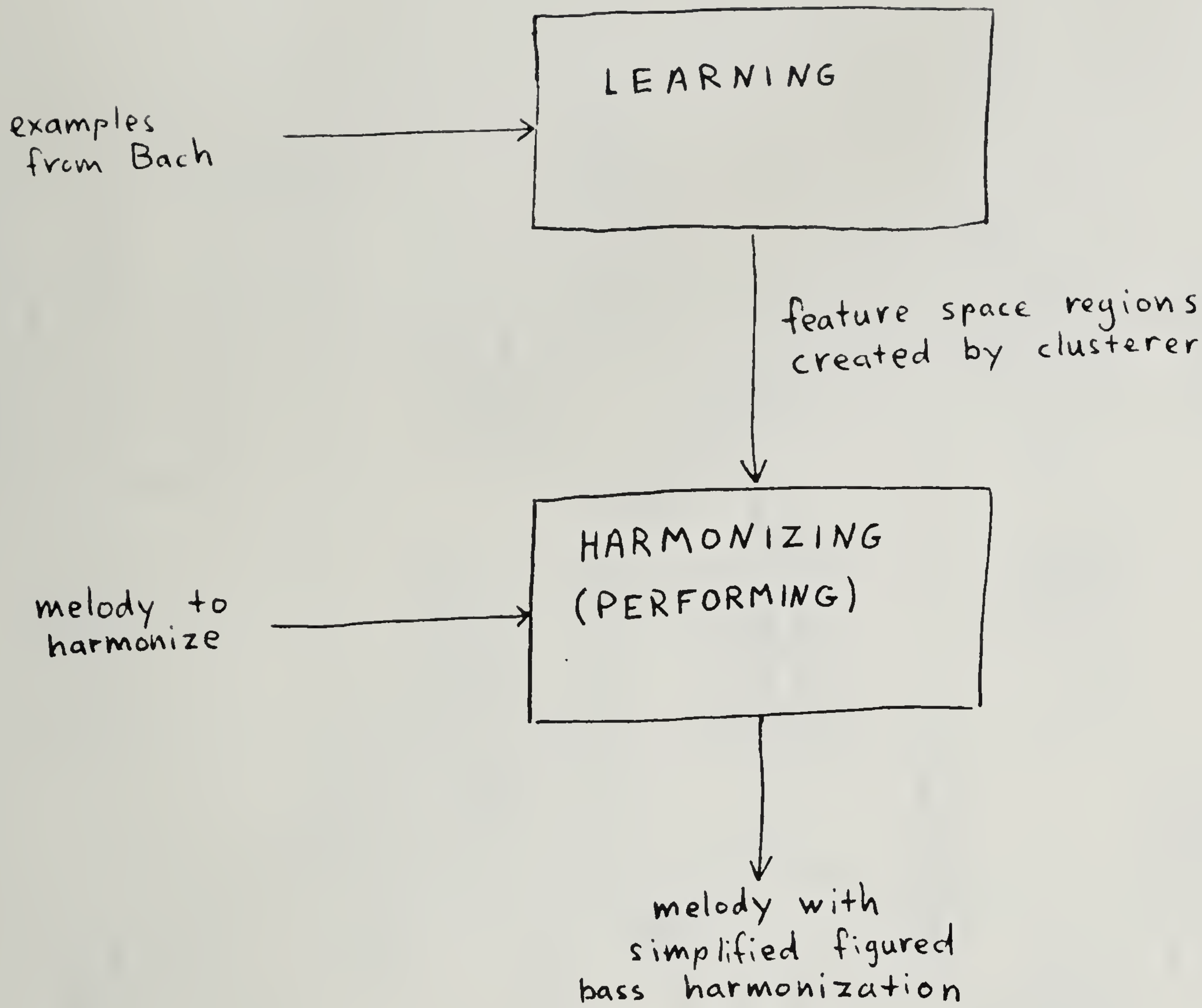


Figure 3. Summary diagram of ChoraLearn.

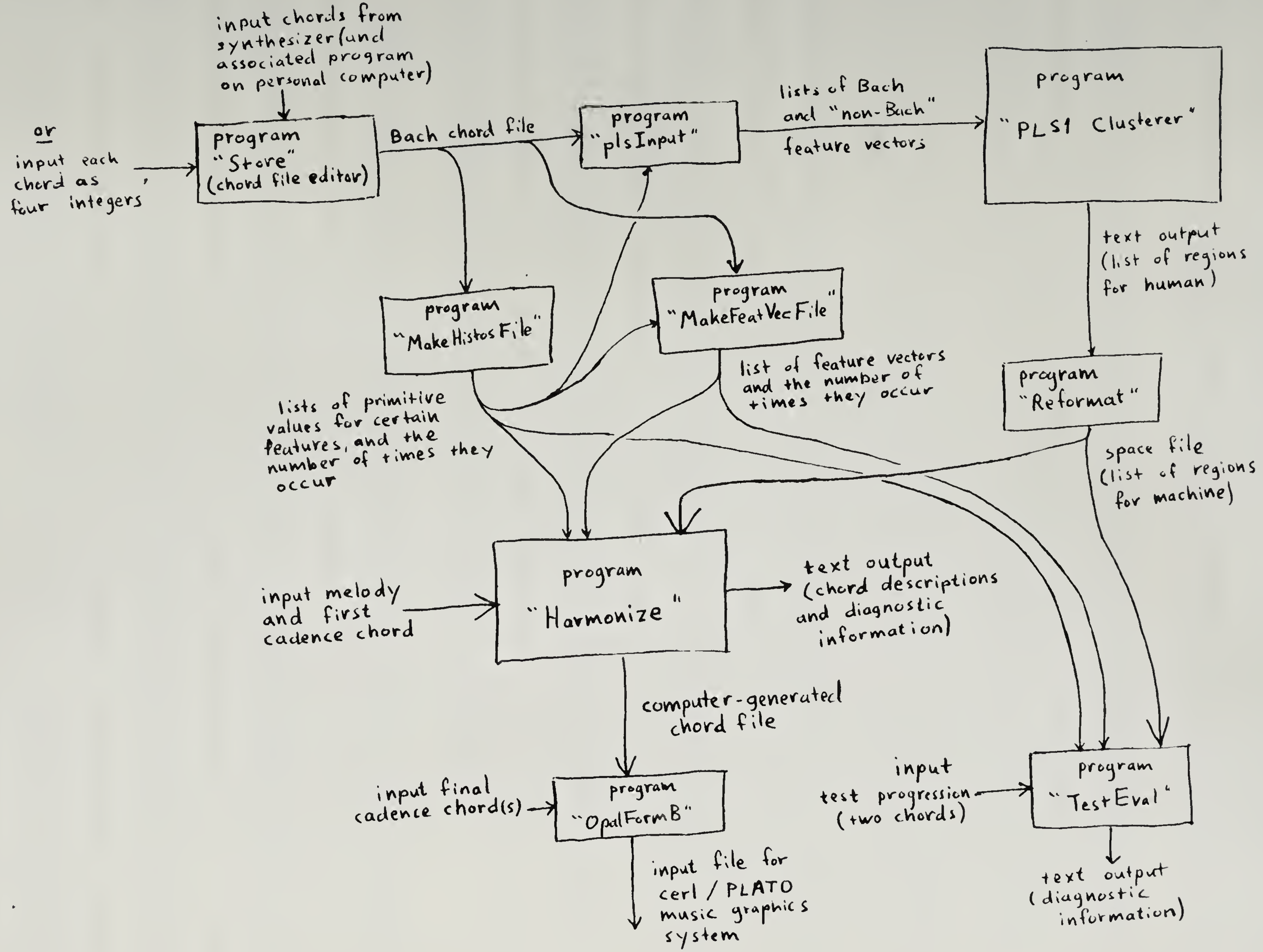


Figure 4. More detailed diagram of ChoraLearn.

objects, but rather as *non-Bach* objects.[6]

[The rest of this section explains in more detail (for the most part musical detail) how the Bach and non-Bach collections were created, and may be skipped without loss of continuity.]

Phrases were chosen from chorales at the beginning of the Riemenschneider (1941) collection of four-voiced chorales. The phrases had to satisfy the constraint that they lie within the problem domain, i.e., they had to be in a major key and not modulate. Each phrase, from first chord to first cadence chord, was input to a chord file editor as a sequence of four-voice quarter-note chords (the key was also input). The quarter-note chords were obtained by my reducing each phrase to a form similar to that shown in Fig. 1d, the only difference being that instead of the alto and tenor notes being as close to the soprano note as possible, they would be the notes that Bach actually wrote. For the most part this reducing process consisted simply of disregarding nonharmonic tones, but in many cases there was eighth-note movement such that the first and second halves of the beat could be analyzed as two separate chords. In such cases I had to decide which was the more important chord. Whenever this decision was difficult, I chose the chord of the first half of the beat.

The four notes of each chord were represented simply as numbers, which were either typed in or played in (via a synthesizer-computer interface). The chord file editor stored the key information as well as a description of each chord in the phrase (the description included the actual input values for the four notes as well as figured bass information extracted from them: type of triad or seventh chord, and inversion). The chord file editor also screened the input. First of all, only triads and seventh chords were accepted. The editor would prompt me to complete ambiguous chord descriptions. For example, is a particular seventh chord

[6]One of the nice things about the PLS1 clusterer is that it can make a useful classification scheme even when it is given collections of "good" and "bad" objects that contain some wrongly classified objects.

with a missing third based on a major or minor triad? Or, is a particular triad with a missing fifth to be analyzed as diminished or minor? A second screen (called MajorKey Screen) accepted only those chords that were in the key. In addition to triads and seventh chords built on the seven scale degrees, those chords included secondary dominant chords and all diminished seventh chords. MajorKeyScreen does not accept second inversions of triads (six-four chords). It also rejects any chord that has the leading tone in both soprano and bass, and any inversion of a seventh chord if the soprano doubles the bass. In the first 23 chorales of the Riemenschneider collection, among the phrases that are in major keys and do not contain modulations (except possibly at the cadence), two had to be rejected because they contained secondary VII_6 chords, and one had to be rejected because it had a noncadential six-four chord for a quarter-note harmony (there were also two six-four chords in chorale 7, and two in chorale 11, each on the second beat of a 3 beat measure, which I just skipped over). MajorKeyScreen is thus a bit too restrictive.

After the file of Bach phrases was prepared using the chord file editor, a program called Histos (not shown in Fig. 4) read through the file, examining each chord progression, and made tabulations of the number of times each value of certain features occurred (the features will be explained in Section B). These tabulations were used in some cases to set bounds on feature space dimensions,[7] and in some cases for grouping and/or reordering of values of nominal features (this was done by hand, but the latest version of the PLS clusterer reorders nominal feature values automatically on the basis of frequency of occurrence). A related program, MakeHistosFile, stored the same tabulations in a file that could be read by the system (programs plsInput and Harmonize—see Fig. 4) for automatic conversion of primitive measures to feature values (for certain features).

[7]To help the clusterer, for some features, values that never occurred in the Bach training examples were made "out of bounds" (see Section B).

The program plsInput read the Bach chord file again to create an input file for the clusterer. The input file consisted of two lists of feature vectors, a "Bach list" and a "non-Bach list." These lists were constructed by the program as follows. For each object in the file (recall that an object, or progression, consists of two successive chords, "Chord" and "NextChord") the program calculated the feature vector and added it to the Bach list. Then it generated all other possible MajorKeyScreened chords (about 60) that had the same soprano note as did Chord of the Bach progression. Each of these, considered together with NextChord, was a new object, and its feature vector was also calculated. If this feature vector was within the bounds of the feature space, it was added to the non-Bach list. (As stated above, the feature space bounds were determined from the preliminary tabulations obtained from the chord file. Of the roughly 60 computer-generated objects corresponding to a single Bach object, the number of objects that were in bounds was on the average 4 or 40, depending on which set of features was used—see below, Section IV.B.)

The clusterer was run and it output a list of feature space regions with associated utilities (see Introduction). Another program, Reformat, stored this list in a form in which it could be used by the performance element of ChoraLearn.

2. Performing (harmonizing)

The performance element took as inputs a melody with a first cadence chord (Fig. 1e). It began to write a figured bass for the melody notes, starting with the last note (the note immediately preceding the first cadence chord), in the following manner (see Fig. 5 for a cartoon of the method). Trial objects were generated corresponding to every possible MajorKeyScreened chord containing the last melody note. Each object consisted of one of the trial chords (Chord) and the first cadence chord (NextChord). The feature vector was then calculated for each of these objects. If it was out of the feature space bounds, then it was

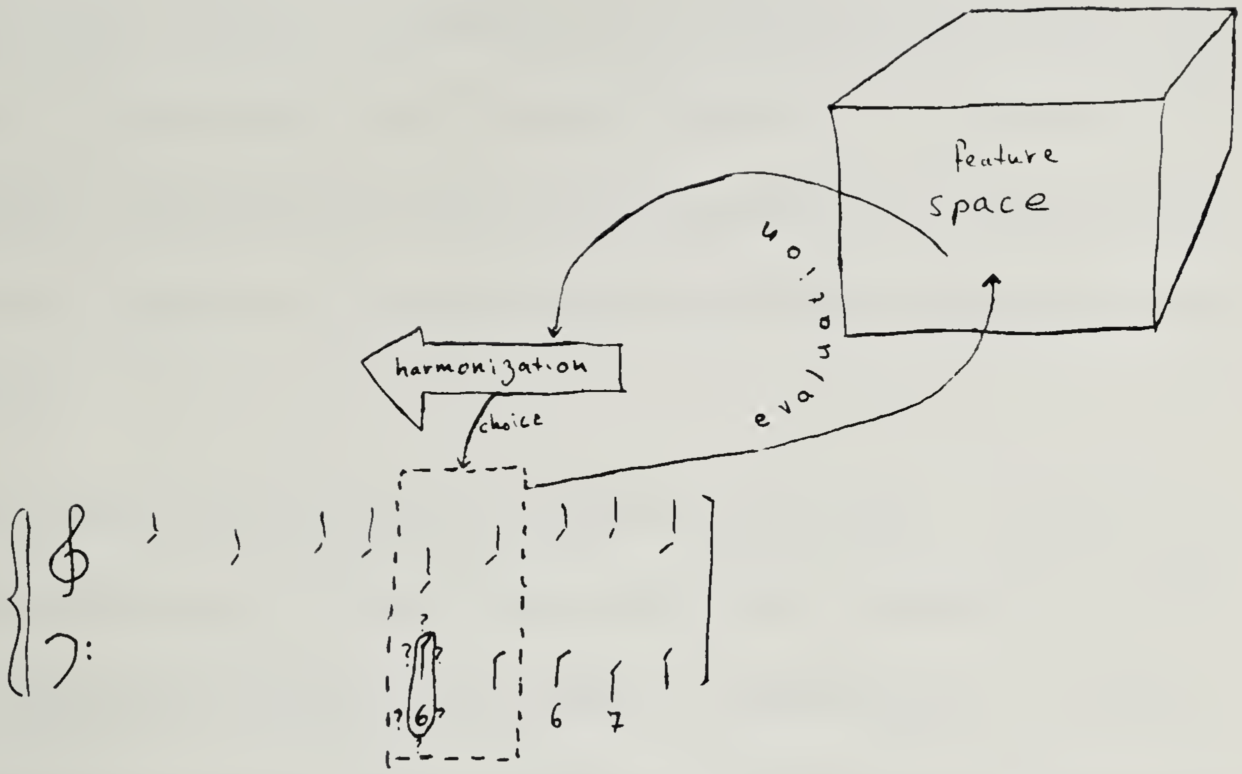


Figure 5. Cartoon of performance (harmonization). The object surrounded by a dashed line is a progression, that which is being evaluated. The smaller object, surrounded by question marks, is a harmony for the note above it, and is what is being chosen.

given a utility of zero. If it was in bounds, then it was given the utility of the region into which it mapped. (The regions for the in-bounds objects were found simply by going sequentially through the region file until the region containing the object's feature vector was found. For improved efficiency, the file was ordered so that the most frequently encountered regions were at the beginning.) After one of the positive-utility progressions was chosen (see next paragraph), the newly determined Chord became NextChord for the next set of progressions to be evaluated, namely those having Chords containing the next to last melody note as the soprano note. This whole process was iterated until Chord contained the first melody note of the phrase.

ChoraLearn can generate harmonizations for a phrase in two modes: "tree mode," in which every possible harmonization constructed with positive-utility progressions (objects) was generated with a depth-first algorithm (Rich, 1983), and "choosing mode," in which a single such harmonization was generated, with the choice for each progression being made completely randomly, or randomly with a weighting formula, from among the positive-utility objects. In the choosing mode, if there were no objects with positive-utility (a dead end), ChoraLearn would "back up" (move towards the cadence chord) three notes (or until it reached the note preceding the first cadence chord if the latter were less than three notes away) and reharmonize from that point. Dead ends occurred more often with some sets of features than with others.

In addition to the two doubling rules contained in MajorKeyScreen (Section 1 above), the performance element contained one other explicit rule: "do not accept a sequence of 5 bass notes with none more than a whole step from the first." The purpose of this rule was to screen out some boring bass contours. When a second stage of learning is implemented, in which features of a measure or whole phrase are used (see above), this rule will be superfluous

(moreover, some Bach bass lines violate this rule).

B. Features

Rendell (1986b) has analyzed systems that use the PLS1 clusterer to learn how to play checkers or solve the fifteen puzzle. He estimates that of the knowledge that is represented in a feature space after clustering, less than half lies in the clustering itself, the rest being in the feature definitions.

The choice of features is therefore very important. The following features were used to construct the feature sets that were used in the tests that will be discussed in the next section. The explanations are mainly for the reader who has some knowledge of harmony. For the musically naive reader, it is sufficient that she keep in mind that the features are used to describe objects that are progressions. Each feature, with the exception of Features 7 and 8, is a basic part of the musical description of Chord or NextChord, or else describes the change in pitch of the soprano or bass in going from Chord to NextChord. Feature 8 is sort of a cheat—a rule disguised as a feature. Feature 7 is also sort of a cheat, being a check of whether a partial specification of the progression, called the *harmonic progression,* occurred in the training examples. Because use of Feature 7 amounts to a partial look-up of the progression, it limits the range of progressions that can be induced as being Bach-like. Feature 8 was included in every feature set, while Feature 7 was omitted from some of them, including the currently most advanced one.

1. Feature 1: BassInterval

The BassInterval is defined as the movement of the bass from Chord to NextChord, i.e., the difference NextBassTone – BassTone (measured in half-step units). This feature had 17 values, corresponding to the 17 different BassIntervals found in the set of training examples

(the complete set of training examples contained 301 chords, but some tests were done using only half of these, as will be described later). The BassIntervals were ordered approximately from greatest descending interval to greatest ascending interval, but with a couple of adjustments made on the basis of the frequency of occurrence (in the training example set) of a particular BassInterval. The rationale for these adjustments was that frequently occurring skips should be located close to the smallest size BassIntervals (-2 to 2) since they account for most of the training examples. The resulting order is shown in Fig. 6. The intervals listed on the bottom, all given the feature value of 20, did not occur in the training examples (downward skip of octave, major seventh, major sixth, upward skip of tritone, major and minor sixth, major and minor seventh). Any trial object having a value of 20 for Feature 1 was out of the feature space bounds (see Section A above).

2. Feature 2: BassRegister

Bach's bass range was from 1 to 27 (in my implementations, 25 is middle C and 1 is two octaves below it). This range was divided into five registers (five feature values). In later feature sets only two registers (feature values) were defined: very low (1 to 5) and everything else.

3. Feature 4: SopranoRegister

The soprano range (25 to 45) was divided into 4 segments.

4. Feature 5: SopranoInterval

The SopranoInterval was defined similarly as the BassInterval (Section 1 above), but the intervals were grouped to form 7 feature values: rarely or never occurring upward skips, frequently occurring upward skips, upward skips, no movement, downward skips, frequently

```
procedure CalcFeat1; {BassTone,NextBassTone:integer;
                                              {var Feat1:DimValType}
  var BassI:integer;
  begin
    BassI:=NextBassTone-BassTone;
    if (abs(BassI)>12)
      then Feat1:=20
      else case BassI of
             -10: Feat1:=1;
              -8: Feat1:=2;
              -6: Feat1:=3;
              -7: Feat1:=4;
              -5: Feat1:=5;
              -4: Feat1:=6;      {CURRENT VALUES FROM   }
              -3: Feat1:=7;      {Bach.12 (303 objects)}
              -2: Feat1:=8;      {      9-1-86          }
              -1: Feat1:=9;
               0: Feat1:=10;
               1: Feat1:=11;
               2: Feat1:=12;
               3: Feat1:=13;
               5: Feat1:=14;
               4: Feat1:=15;
               7: Feat1:=16;
              12: Feat1:=17;
             -12,-11,-9,6,8,9,10,11: Feat1:=20
           end;
  end; {CalcFeat1}
```

Figure 6. Code for Feature 1.

occurring downward skips, rarely or never occurring downward skips.

5. Feature 6a: BassScaleElement

The BassScaleElement was one of the twelve degrees of the chromatic scale (tonic = 1, leading tone = 12). These were ordered according to their frequency of occurrence in the set of Bach training examples (Fig. 7). Degree 4 (lowered third degree) did not occur in the training examples (which were all in major keys) and was given the value 20, putting it out of feature space bounds.

6. Feature 7: HarmonicProgression

The HarmonicProgression was defined as the sequential harmonic analyses of the Chord and NextChord. The harmonic analysis of a chord was in turn uniquely specified by the BassScaleElement, the type of triad (major, minor, or diminished), the inversion, and the kind of seventh (major or minor third higher than the fifth), if there was one. This information was represented in a four-digit code.[8]

To assign a feature value to an object, the HarmonicProgression was calculated and looked-up in a table containing all the HarmonicProgressions of the training examples. Frequently occurring HarmonicProgressions were given a value of 1, infrequently occurring harmonic progressions were given a value of 2, and those not in the table were put out of feature space bounds. Inclusion of this feature thus restricts the range of progressions that can be induced, and it was therefore not included in the most advanced feature set (see Section IV.G).

[8]The entire job of ChoraLearn, as stated above, is to learn what constitutes a good progression. A progression, i.e., the object consisting of Chord and NextChord, consists of the HarmonicProgression plus the two soprano notes.

```
procedure CalcFeat6a (Key,BassTone: integer; var Feat6a: DimValType);
 {frequency of occurrence of bass note scale element}
 var BSE: integer;
 begin
   BSE:=(((BassTone+12)-(Key mod 12)) mod 12)+1;
   case BSE of
     1: Feat6a:=1;  {most common}
     5: Feat6a:=2;
    10: Feat6a:=3;
     6: Feat6a:=4;
    12: Feat6a:=5;
     8: Feat6a:=6;
     3: Feat6a:=7;      {CURRENT VALUES FROM  }
     7: Feat6a:=8;      {Bach.12 (303 objects)}
     9: Feat6a:=9;      {    9-1-86           }
    11: Feat6a:=10;
     2: Feat6a:=11;
     4: Feat6a:=20
   end
  end; {CalcFeat6a}
```

Figure 7. Code for Feature 6a.

7. Feature 8: parallel and hidden fifths and octaves

Feature 8 is the only feature that took the form of a rule. If an object had parallel fifths or octaves between the soprano and the bass, it was assigned a Feature 8 value of 3. If it had a hidden fifth or octave between soprano and bass it was assigned a Feature 8 value of 2.[9] Otherwise it was assigned a Feature 8 value of 1. None of the objects in the Bach training examples had a Feature 8 value of 3, so this value was made out of feature space bounds. There were, however, two cases of Feature 8 value 2 in the training examples (both hidden fifths). A nice result of clustering, then, would be if objects having a Feature 8 value of 2 were evaluated to have a positive utility only if they were similar to the two Bach examples.

8. Features 9, 10, and 14: SopranoChordElement and BassChordElement

Features 9 and 10 specified the chord element (root, third, fifth, seventh) of the soprano and bass respectively of Chord. Feature 14 specified the chord element of the bass of NextChord.

9. Features 11 and 13a: Triad

Feature 11 specified whether Chord was based on a major, minor, or diminished triad (augmented triads were screened out by MajorKeyScreen). Feature 13a did the same for NextChord.

10. Feature 12: Seventh

This feature specified whether Chord was a seventh chord, and if so, whether the seventh degree was a major or minor third above the fifth.

---

[9]Piston's (1962) definitions were used: a hidden fifth is a fifth approached by similar skipwise motion; a hidden octave is an octave approached by downward skipwise motion in both voices, or by upward motion of both voices (except if soprano moves upward by half-step and bass by skip).

IV. CHORALEARN: TESTS AND RESULTS

In this section some of the results obtained using ChoraLearn will be discussed. The feature sets used were as follows (see previous section for explanations of individual features):

TABLE 1

COMPOSITION OF FEATURE SETS

Feature Set	Features
2	1, 2 (5 values), 4, 5, 6a, 7, 8, 9, 10
3	1, 2 (5 values), 4, 5, 6a, 8, 9, 10, 11, 12
4	1, 2 (2 values), 5, 6a, 7, 8, 9, 10
5	1, 2 (2 values), 5, 6a, 8, 9, 10, 11, 12
6	1, 2 (2 values), 5, 6a, 8, 9, 10, 11, 12, 13a, 14

[Feature Set 1 was used in very preliminary tests, and will not be discussed.]

A. Choosing a Progression: "OKChord," "GoodChord," or "BestChord"

Up to the present state of development, ChoraLearn has been learning how to evaluate, or classify, a single progression (the object consisting of Chord and NextChord). It can string progressions together, choosing those that it classifies as Bach-like, to create a harmonization for the noncadential portion of a phrase. The objective has been to be able to generate, for a given phrase, a set of such harmonizations that is likely to include some that are overall Bach-like (see above, Section II.C.2).

One set of harmonizations that the present ChoraLearn can generate is simply the set of all possible harmonizations using progressions with positive-utility (the "tree-mode" mentioned in Section III. A. 2). In this case utility is reduced to a boolean function (this is not always done in other applications of the clusterer).

The number of possible ways to harmonize a phrase with k noncadential melody notes is N^k, where N is the number of positive-utility progressions for a given melody note and

NextChord. (Using the melodies of the training examples and Feature Set 6, I obtained, with the help of a program, an average value of 3.6 for N .) Thus, for long phrases, the size of the set of all possible harmonizations is unmanageably large (e.g., if k is 10, the estimated number of possible harmonizations is about 4×10^5).

There are several ways to generate a smaller set of harmonizations. I will discuss now only two of the ways that have been implemented in ChoraLearn. One way was simply to choose at random one of the positive-utility progressions for each successive melody note. A harmonization generated this way I called an "OKChord" harmonization. ChoraLearn could be asked to generate as many OKChord harmonizations as desired to create a set of arbitrary size. The other way was similar, except that the random choice was weighted such that the probability of a given progression being chosen was proportional to its utility divided by the sum of the utilities of all the possible progressions (for the given soprano note and NextChord). A harmonization generated this way was called a "GoodChord" harmonization.

[The rest of this section is for the reader interested in machine learning details, and may be skipped without loss of continuity.]

In considering the relative merits of a GoodChord harmonization and an OKChord harmonization, one must recognize that some potentially important information is lost in reducing utility to a boolean function. Recall from Section I.C that the utility of a region is the number of good objects divided by the total number of objects in the region. In ChoraLearn, the feature space was filled with Bach progressions and all possible alternatives to them (see Section III.A.1). Thus, in our system, the utility of a region is the number of times Bach used a progression from that region divided by the number of times he *could have*

used one in that region (in the training example situations).[10] Thus it makes sense for ChoraLearn to preferentially choose progressions with high utilities. Of course, we could also have made this choice deterministic, i.e., chosen the highest utility progression every time, but there would be only one such "BestChord" harmonization. We know that we need to generate a *set* of harmonizations because the choice of a good harmonization will require examining global as well as local features (the second stage of learning, mentioned above).

Preliminary tests with two fragments of chorale phrases (not included in the training examples), using a program that calculated the average utility of the chords in each possible harmonization, showed (in both cases) that the Bach versions had average utilities that were near the highest possible. The GoodChord choosing procedure also produced versions with high average utilities, sometimes reproducing the Bach version. The BestChord version had in one case the highest average utility, and in the other, the second highest (in neither case was it the same as the Bach version).

B. Pruning the Set of Possible Choices for a Progression: Examples Obtained with Two Different Feature Sets

In this short section we will get an idea of the actual sizes of the sets of available progressions as they are successively reduced by the screening operations of ChoraLearn (see Fig. 2). The final screen is the classification according to feature space regions. The next to final screen is the feature space bounds. The amount of progressions it screens out depends very much on the feature set. In particular, if Feature 7 is included in the set, the bounds are very restrictive (they screen out a lot of progressions). This leaves less work for the clusterer

[10] Following this line of thinking, one might think it possible to extract rules from those regions that had a 1.0 utility, because Bach used a chord from such a region every time he could (in the training examples). In fact, in the results I have examined, there were at most two such regions in any clustering of a space containing 150 to 300 Bach objects, and each of these regions had only 5 or 6 objects in it. Furthermore, the 5 or 6 points were not always closely related, and the regions were very dependent on the feature set and the number of training examples. Such regions thus seemed rather to represent fortuitous interactions of feature sets and training examples than consistent practices of Bach.

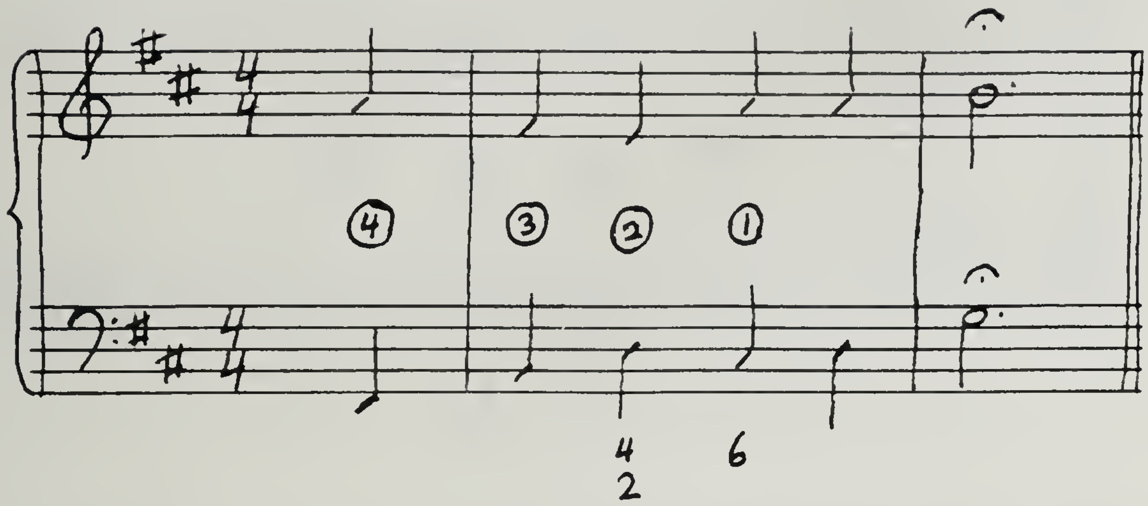
to do, but the disadvantage of not allowing the system to generalize as much (see Section III.B.6) is a real one, as will be seen in the next section (see also footnote in section D below).

[The rest of this section gives representative numbers for the sizes of the sets of available progressions, and elaborates further on the important differences between feature sets that include Feature 7 and those that do not. It may be skipped without loss of continuity.]

An example of an OKChord harmonization using Feature Set 2 is shown in Fig. 8. Let us consider the fourth text entry below the music. It contains some information about how Chord no. 4 was chosen. The total number of trial Chords generated was 294. Of these, 63 passed the MajorKeyScreen (see Section III) for the key of D. Of these 63, only 6 were in the feature space bounds, and of these 6, 3 were in regions having positive utility ("good" regions). The chord chosen had a utility of 0.3636 ("Eval" is the utility multiplied by ten thousand). In going from the MajorKeyScreened chords to the positive utility chords, the problem has been reduced from choosing one chord out of 63 to choosing one chord out of 3.

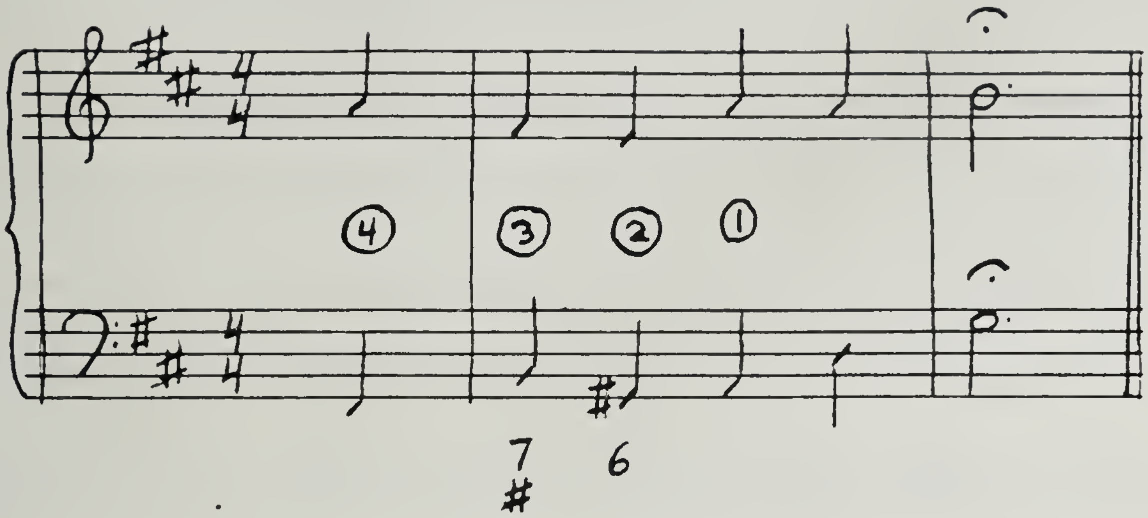
Figure 9 shows a similar output as in Fig. 8. However, it comes from a trial in which Feature Set 3 was used. Looking at the text entries, we see that there is a big difference in the number of objects that were in feature space bounds. The number in bounds in the first entry, 38, is typical.

The major difference between Feature Sets 2 and 4, on the one hand, and Feature Sets 3, 5, and 6 on the other, is that the former include Feature 7, which refers to the list of HarmonicProgressions found in the training examples, whereas the latter rely on Features 11, 12, 13a, and 14, which, together with some of of the other features, contain the information




```
Chord chosen is no. 4 of 63 MajorKeyScreened chords.
Total no. of chords generated was 294
no. in Bach feature space bounds: 13              (1)
no. in "good" clustered regions: 3
 11   3   2   5   5   1   1   1   2        Eval= 8000

Chord chosen is no. 36 of 55 MajorKeyScreened chords.
Total no. of chords generated was 231
no. in Bach feature space bounds: 7               (2)
no. in "good" clustered regions: 4
  9   3   1   3   1   2   1   1   4        Eval= 2500

Chord chosen is no. 47 of 69 MajorKeyScreened chords.
Total no. of chords generated was 251
no. in Bach feature space bounds: 1               (3)
no. in "good" clustered regions: 1
 13   2   2   6   3   2   1   3   1        Eval= 6667

Chord chosen is no. 40 of 63 MajorKeyScreened chords.
Total no. of chords generated was 294
no. in Bach feature space bounds: 6               (4)
no. in "good" clustered regions: 3
 14   2   2   7   2   1   1   2   1        Eval= 3636

phrase written to file C291a02
cpu time was  5.53300000000000e+00 seconds
```

Figure 8. Example of a harmonization generated with Feature Set 2.




```
Chord chosen is no. 1 of 63 MajorKeyScreened chords.
Total no. of chords generated was 294
no. in Bach feature space bounds: 38           ①
no. in "good" clustered regions: 4
 14  2  2  5  6  1  1  1  1  0     Eval= 1667

Chord chosen is no. 3 of 55 MajorKeyScreened chords.
Total no. of chords generated was 231
no. in Bach feature space bounds: 33           ②
no. in "good" clustered regions: 2
 11  2  1  3  8  2  1  2  1  0     Eval= 4000

Chord chosen is no. 27 of 69 MajorKeyScreened chords.
Total no. of chords generated was 251
no. in Bach feature space bounds: 49           ③
no. in "good" clustered regions: 7
  7  2  2  6  3  1  3  1  1  1     Eval= 2000
progression not in Bach file

Chord chosen is no. 40 of 63 MajorKeyScreened chords.
Total no. of chords generated was 294
no. in Bach feature space bounds: 38           ④
no. in "good" clustered regions: 4
 14  2  2  7  2  1  2  1  2  0     Eval= 3750
progression not in Bach file


phrase written to file C291a08
cpu time was  2.69330000000000e+01 seconds
```

Figure 9. Example of a harmonization generated with Feature Set 3.

to *partially* specify a HarmonicProgression.[11] With Feature Sets 2 and 4, in order for a trial object to be in feature space bounds, it had to have a HarmonicProgression identical to the HarmonicProgression of one of the training examples. This is a very restrictive condition, and for this reason the number of in-bounds trial objects was so small with Feature Sets 2 and 4. The greater amount of work done by the clusterer with Feature Set 6 (lacking the restrictive Feature 7) than with Feature Set 4 is seen in the number of non-Bach training objects (11,802 vs. 1,399) and in the cpu time for clustering (62 min vs. 7 min).

C. Some Harmonizations Generated by ChoraLearn Using Feature Set 6

In this section and the next we will concentrate on the system's performance using Feature Set 6, the most advanced of the feature sets, and a set of 301 Bach training examples. Here I present some computer-generated harmonizations, and in the following section I present an evaluation, partly based on them, of the learning done by the clusterer. In this section we will see that, at least for a short phrase, a small set of harmonizations generated by ChoraLearn has a good chance of containing some that are overall Bach-like.

We asked the computer to generate all possible harmonizations (using progressions that map into positive-utility regions) for the first phrase of chorale "O Gott, du frommer Gott."

The results are shown in Fig. 10. In addition to the figures below the bass notes, which are the figures of the figured bass, there are also some numbers and letters in between the treble and bass staves. The numbers mark the progressions that were induced, i.e., the progressions whose feature vectors are not represented in the Bach training examples. There

[11]The components of the HarmonicProgression specification that cannot, in general, be found from a feature vector in Feature Set 3 or 5 are three of the four that constitute the harmonic analysis (see Section III. B. 6) of NextChord. Only the BassScaleElement of NextChord is computable from the BassScaleElement of Chord and the BassInterval (going from Chord to NextChord). In Feature Set 6, Features 13a and 14 specify the NextChord triad and bass chord element respectively, leaving only the NextChord seventh unspecified. The fact that the HarmonicProgression is not completely specified by the features in Feature Sets 3, 5, and 6 did not affect the number of trial objects that were in feature space bounds.

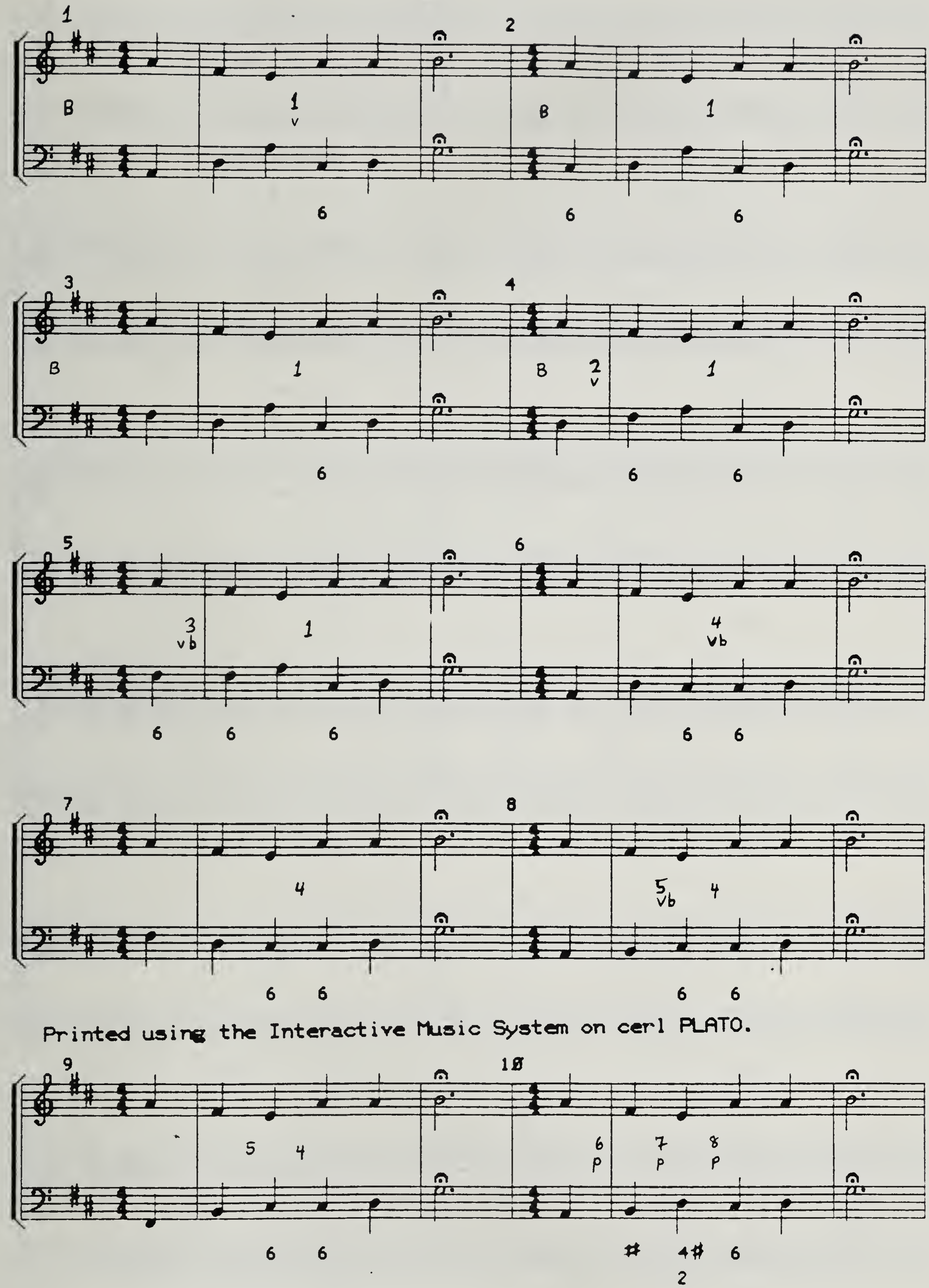


Figure 10. All possible harmonizations of a phrase generated by ChoraLearn with Feature Set 6 (beginning).

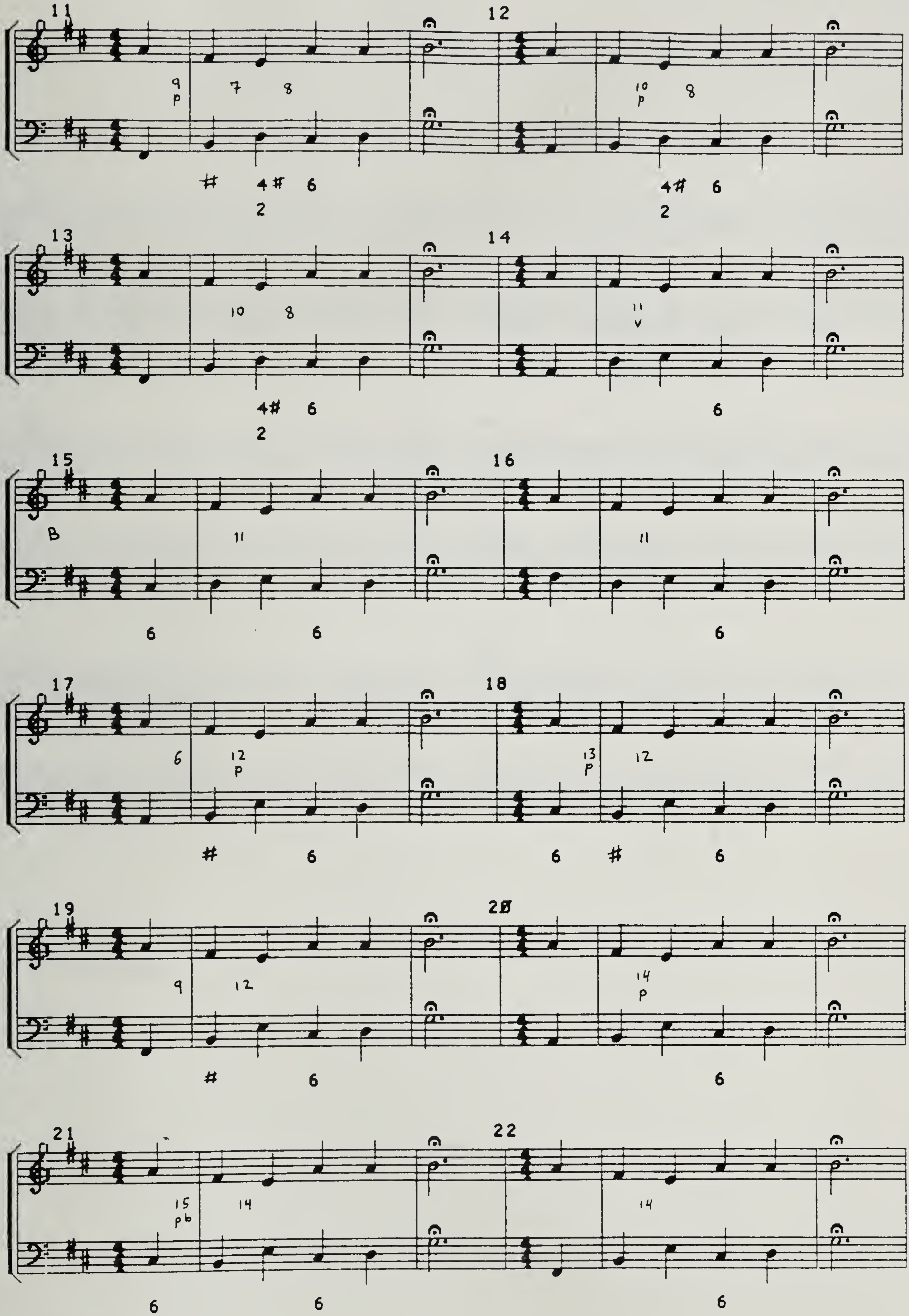


Figure 10 (continued).

Figure 10 (continued).

were 19 such progressions, though many of these occur more than once. The first time one of these progressions occurs it has either a "v" or a "p" underneath it. A "p" indicates that the HarmonicProgression of the progression (the value of Feature 7) did not occur in the Bach training examples. A "v" indicates that the HarmonicProgression did occur in the training examples, although the particular feature vector did not. We asked a musician well-acquainted with the chorales of Bach (hereafter referred to as "the Bach expert") to examine briefly the marked progressions. She said she could imagine that all but five of them might have been used by Bach in some context in a chorale.[12] These five, some of which she considered weak or questionable, are identified by a "b" (for "bad") next to the "v" or "p." It should be noted that the twelve progressions marked with a "p," of which only two were marked bad, could not have been induced using a feature set that included Feature 7.

We also asked the Bach expert to make a quick survey of the set of harmonizations to see if she could imagine that any of the whole-phrase simplified figured bass harmonizations had been written by Bach (actually, the cadence in each case, consisting of the last two chords, was copied from Bach, not generated by computer). She judged five of them to be Bach-like. These are marked with a "B" at the beginning of the phrase.

It is interesting to note that the two harmonizations consisting entirely of progressions whose feature vectors occurred in the Bach training examples (versions 27 and 28) were not among the five that the Bach expert judged to be Bach-like. If we assume that this was not just an oversight of our expert, it illustrates two things: (1) the value of being able to produce progressions similar to, but not identical to, those in the training examples (i.e., being able to *induce* good progressions), and (2) the necessity of eventually being able to look at features of a whole phrase, or at least a measure, of music.

---

[12]In some cases it was difficult for the Bach expert to decide whether Bach might have used a particular progression somehow, somewhere or not.

Another thing to note in Fig. 10 is that the third chord from the end, which was not a cadence chord, is the same in every version. This is because it was the only chord, which, considered together with the first cadence chord following it, mapped into a positive–utility region. This was not the chord that Bach used, and so the Bach version (actually, there are two Bach versions of this phrase, which differ only in the first chord) is not included in Fig. 10. If, however, we replace the third chord from the end by the corresponding chord from Bach's version, then the rest of the progressions (working towards the beginning) in the Bach version all map into positive–utility regions, and so could be generated by ChoraLearn (actually, this is true only for one of the Bach versions -- in the other one, the first progression evaluated to zero). In fact, it appeared as one of the first ten versions generated with the GoodChord choice procedure (see Section B above). These are illustrated in Fig. 11 (the Bach version is number 7). This might not seem very impressive, since the computer is only generating three chords. However, if we assume an average of 40 in–bounds progressions for each choice and were to choose randomly from them, the chances of getting a particular result in 10 trials is about one in 6,500. We tried to see if we could replicate Bach in another case, in which the computer had to generate four progressions, and there were three possible Bach versions. In this case, a Bach version did not occur in the first 10 trials, but did in the first 30 trials. The chance of one of the four Bach versions occurring in 30 trials if the computer chose at random from in–bounds chords is about one in 30,000. These figures indicate that the choices influenced by the clusterer's action are much better than random. In other words, the system has learned something from the Bach examples. Further quantitative evidence for this is presented in the next section.

Figure 12 shows the first 10 GoodChord versions of a phrase from an Indonesian folk song (the fermata in the middle of the line does not represent a cadence). In this case, almost

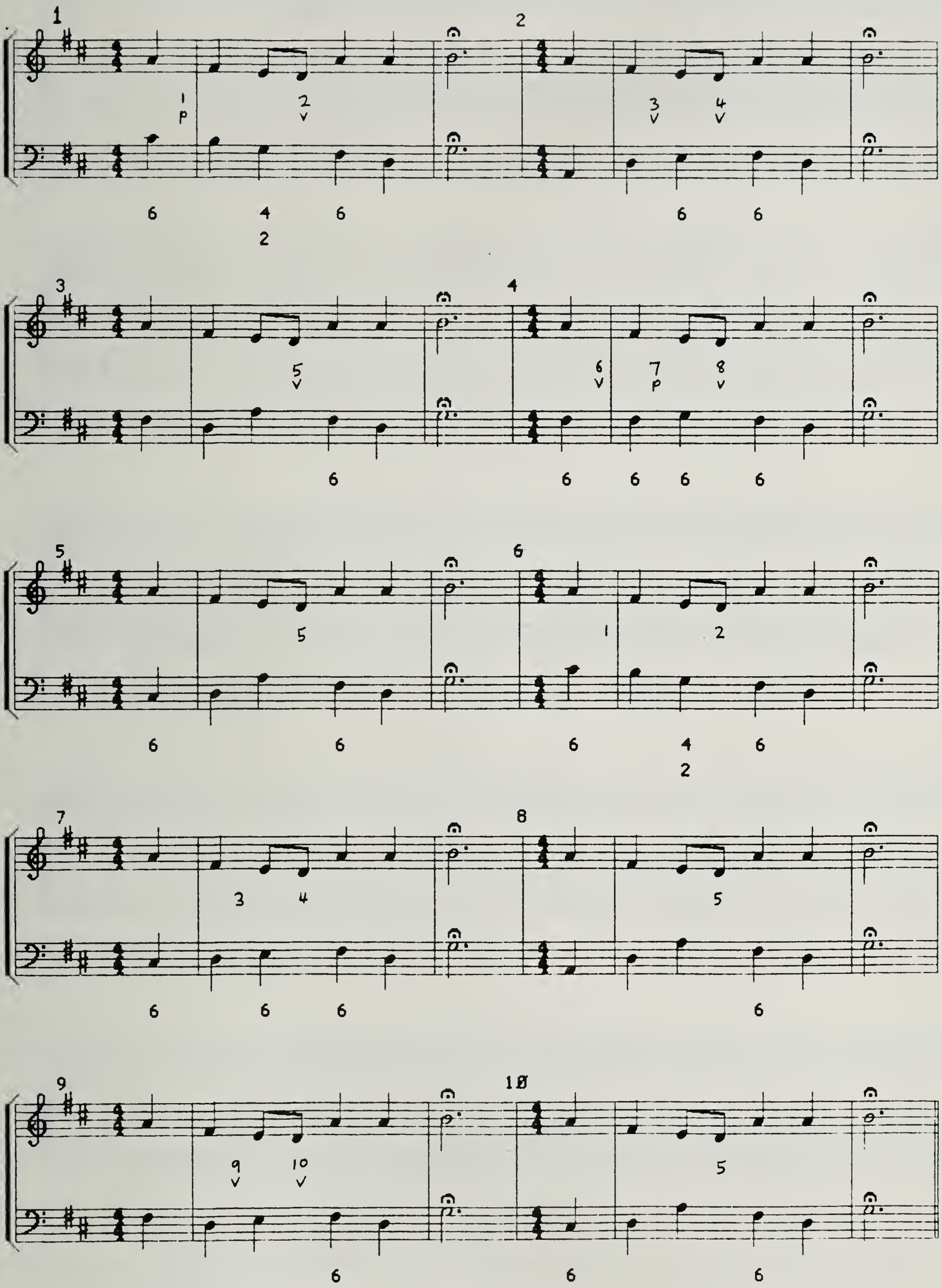


Figure 11. The first ten GoodChord choice harmonizations of the first three notes of the phrase in Fig. 10.

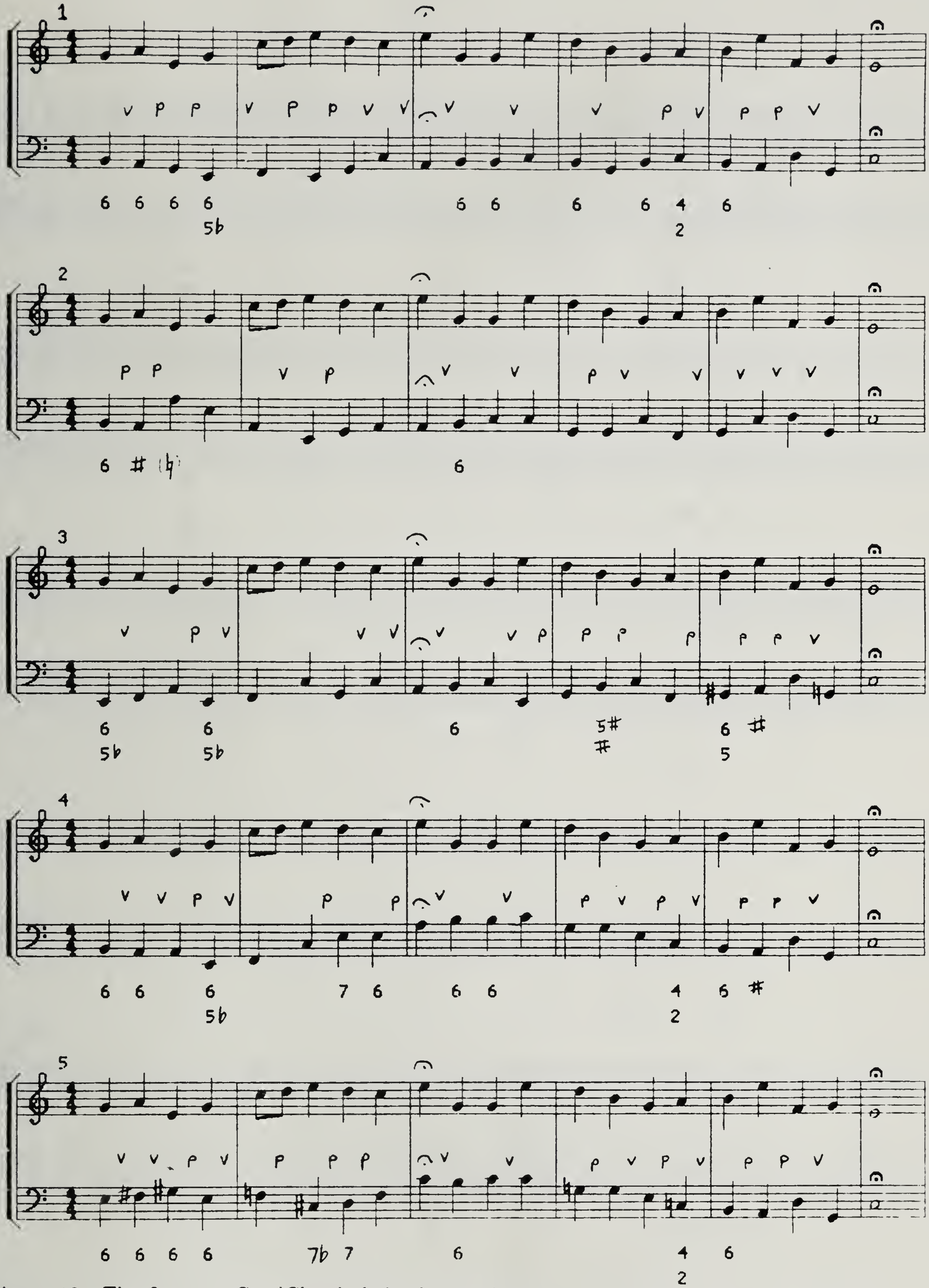


Figure 12. The first ten GoodChord choice harmonizations of a phrase from an Indonesian folk song (beginning).

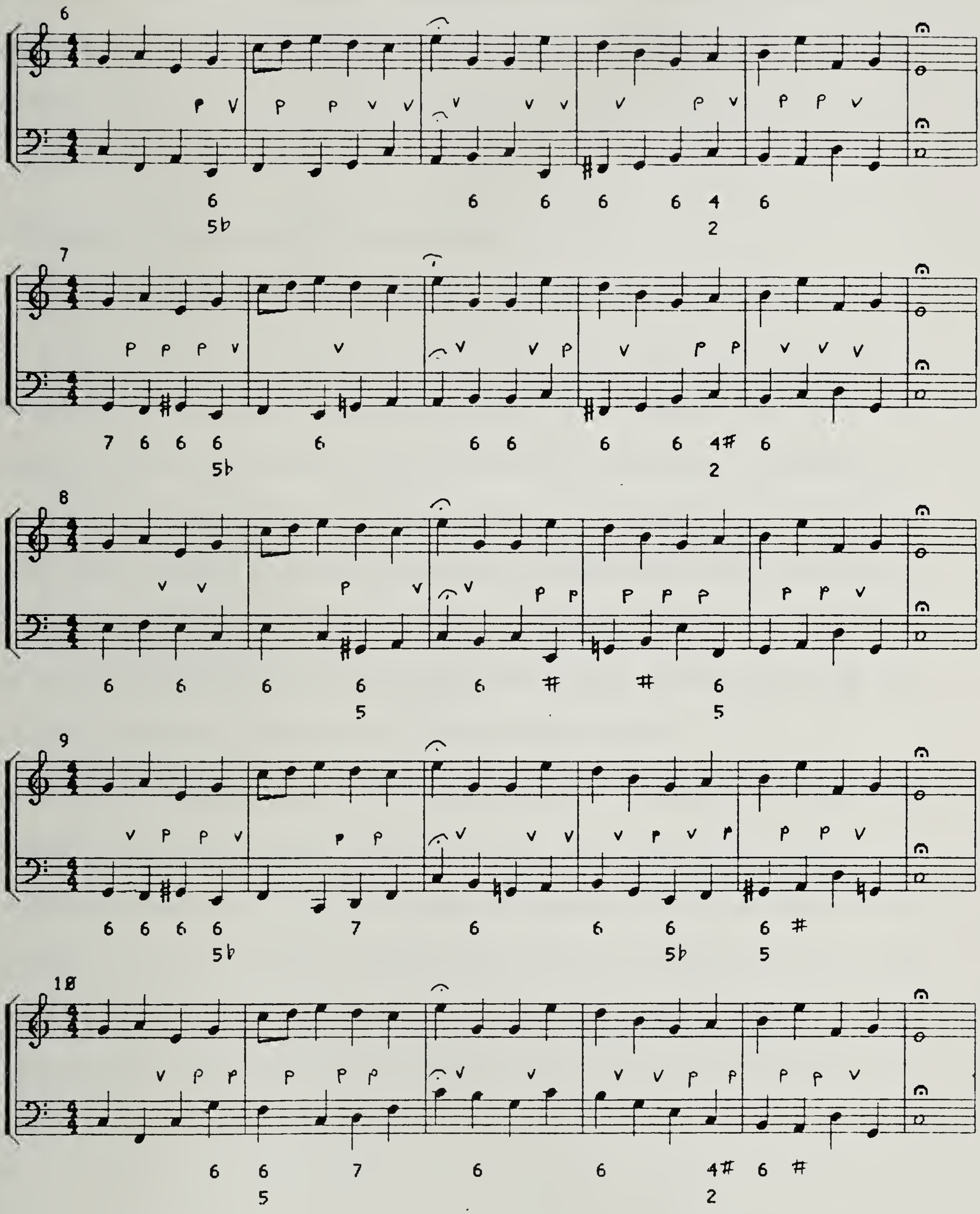


Figure 12 (continued).

every feature vector did not occur in the training examples because the melody is quite different from any of the Bach examples (v and p have the same meanings as in the previous two figures). Nevertheless, some of the figured bass harmonizations generated by ChoraLearn are fairly pleasing to most ears.

D. Evaluation of the Learning Done by the Clusterer

1. The need for two different measures

As stated in Section II.C.2, we would like to measure how much closer the set of Bach–like–according–to–the–clusterer progressions comes to being identical to the set of truly Bach–like progressions than does the set of all in–bounds progressions. A straightforward way to do this is to test the system. The test should consist of two parts. In one part, the system should be allowed to generate objects (two–part progressions). Its performance would be measured by the fraction of these objects that are good (Bach–like). In the other part of the test, the system should evaluate a set of good (Bach) objects. Its performance in this part would be measured by the fraction that it correctly classifies as good.

It is easy to see the necessity of two parts or measures. If a system were judged only according to how few bad objects it generates, then a system could be too restrictive, i.e., capable of generating only a very limited number of progressions, and yet be considered to be very good. On the other hand, if a system were judged only according to how many good objects it is capable of generating, then a system could be not restrictive enough, i.e, capable of generating many good objects, but also many bad objects, and yet be considered to be very good.

2. The test

We conceived a test for ChoraLearn as follows. For the first part of the test, we used the set of 28 possible harmonizations that the system wrote for phrase 1 of "O Gott, du frommer Gott," because these were evaluated for us by a Bach expert (Fig. 10). In these harmonizations, the system generated a total of 32 different progressions. Of these, 13 had feature vectors identical to progressions in the training examples, and so were assumed to be good. Of the remaining 19, 14 were judged to be Bach-like by the expert. Thus 27, or 84%, of the computer-generated objects were good.

Of course, if there were no clustering at all, and the system accepted all objects that were within the feature space bounds, some fraction of these in-bounds progressions would be good. To get an estimate of what this fraction would be, we assume that a completely correct clustering would have approximately the same amount of feature space contained in positive-utility regions as does the current clustering. In the current clustering, 9.1% of the total in-bounds points are in positive-utility regions. Therefore, 9% is a rough estimate of the proportion of good objects that would be generated by a system that accepts any object in the space (no learning). [We could get approximately the same estimate from Fig. 9 by dividing the number of progressions in "good" regions by the number in the feature space bounds, because Feature Set 3 is similar to Feature Set 6.]

For Test II, 30 progressions from Bach chorales not in the training examples were evaluated by ChoraLearn (using the same clustering as for Test I). Of these, 24 (80%) were correctly evaluated as being good (positive utility).[13] Of the remaining six, one had a movement of the bass that was outside feature space bounds, and two others had a chord

[13] Five of the correctly evaluated progressions had HarmonicProgressions that were not in the training examples, and therefore would have had a zero utility according to a system that used a feature set containing Feature 7.

that was not MajorKeyScreened as either Chord or NextChord. Thus, a perfect clustering, within the given constraints, would have resulted in a score of 90%. A nonrestrictive system with no clustering at all, i.e., a system that accepts any object in the feature space, would also get a score of 90%.

Adding the scores from the two tests together, we get the composite scores of 99 (out of 200) for the nonclustered system, 164 for the clustered system, and 190 for a perfect system. Normalizing, we find that the clusterer, with the current set of training examples, has gone 71% of the linear distance from considering everything in the feature space to be good[14] to having learned perfectly which points in the space are good and which are not. We anticipate that the proposed feedback mechanism, which is described in Section V, will enable the system to go most of the remaining distance.

The test scores mentioned above for the current system may not seem very impressive, but in fact they represent learning that has made a very substantial contribution towards the goal of being able to generate Bach-like harmonizations for a phrase. This is explained in the following section.

3. Importance of the learning done so far

In order to appreciate the real significance of the learning quantified above, it is necessary to recall that the goal of the system thus far has been to be able to generate a set of piecewise-Bach-like harmonizations for a phrase, such that a future system will be able to select a smaller set of overall-Bach-like harmonizations using a feature space constructed from more global features than those of the current system.

---

[14] If, instead of taking a completely nonrestrictive system as a zero-learning baseline, we take an almost completely restrictive system, e.g., one that has learned one progression by rote and considers everything else to be bad, we get a similar result. That is because such an "almost no learning" system would score 100 on Test I (assuming that the phrase(s) it were given to harmonize allowed it to use its single learned progression) and zero, or close to zero, on Test II, yielding again a composite score of about 100.

It follows that we want the phrases generated by the present system to have a reasonable probability of being overall–Bach–like. This goal obviously becomes more difficult to achieve as the length of the phrase increases. Figure 13 illustrates abstractly the way the number of possible harmonizations is reduced by the various screens of the system (see Section B above). As the number k of noncadential chords in the phrase grows, so do the relative differences in size between the various regions. I have assumed that the average number of Bach–like progressions is approximately the same as the average number of positive–utility progressions available given a NextChord and a soprano note for Chord when Feature Set 6 is used. (As mentioned earlier, this average is 3.6. Here, I have rounded it off to 4.) The number of overall–Bach–like phrases, 3^k, was roughly extrapolated from the information in Fig. 10 from our Bach expert that for $k=4$, 5 of 20 piecewise–Bach–like phrases were overall Bach–like.

The probabilities of generating overall–Bach–like phrases of different lengths are estimated in Table 2. The column on the far right contains the estimates for a system that has perfectly learned to recognize Bach–like progressions. The column to the left of it is for the current system, and was calculated using the 84% score on Test I mentioned above. I assumed that the imperfection represented by the less than perfect score on Test II, while limiting slightly the variety of possible Bach–like results, would not appreciably affect the proportion of generated phrases that would be overall Bach–like.

Looking at Table 2, we can easily see the great advantage of the current clustering (positive–utility progressions) over choosing randomly from in–bounds progressions. For a phrase with ten noncadential notes the advantage was the difference between a probability of 0.01 for generating an overall–Bach–like phrase versus a probability of 6×10^{-12}, a factor of 2×10^{9}. The importance of improving upon the 84% correctness (measured on Test I) of the

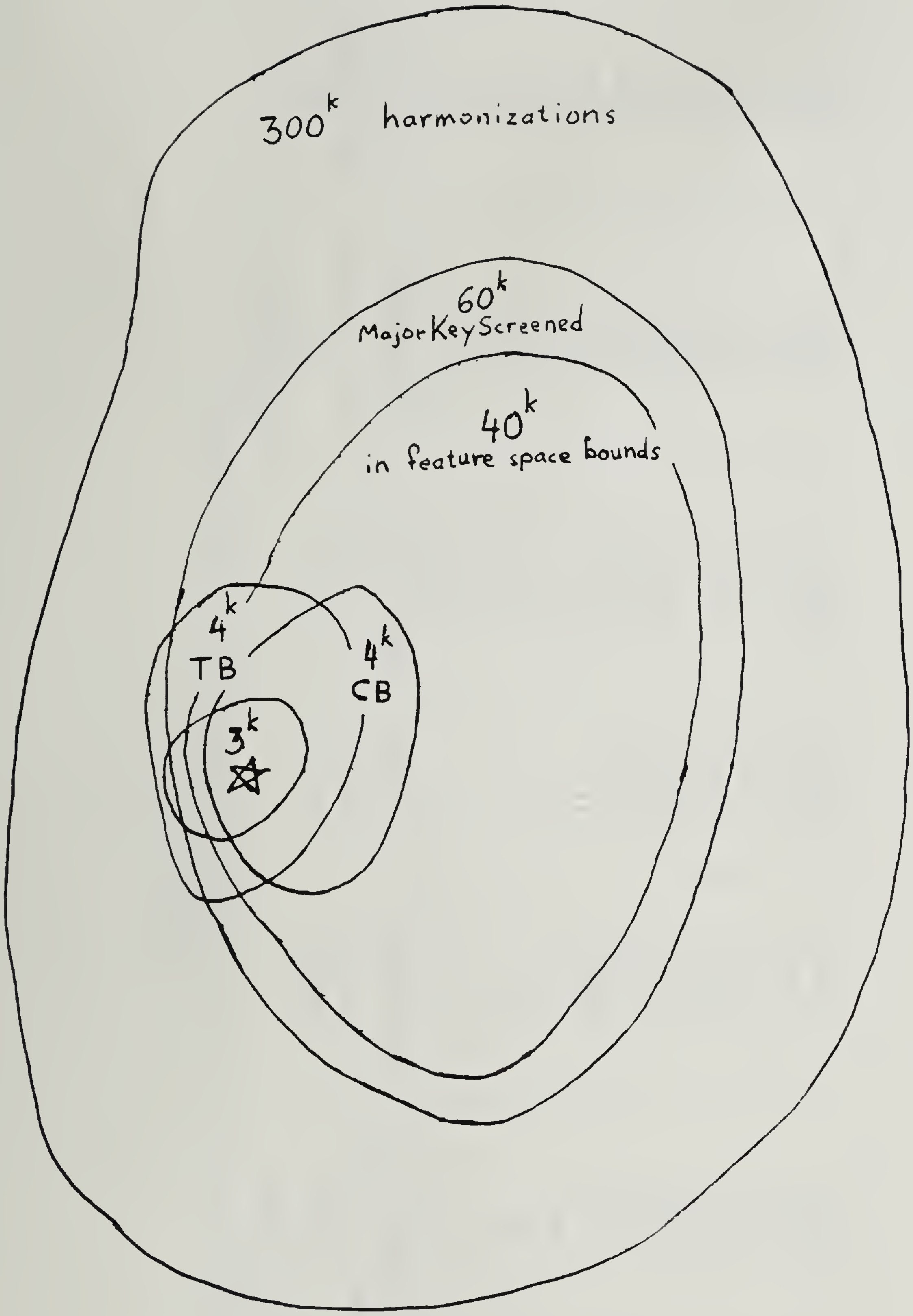


Figure 13. Abstract representation of the sets of harmonizations that could be constructed with different sets of progressions. The exponent k is the number of non-cadential chords in the phrase. The $\check{C}$B region represents the set of harmonizations that are constructed from progressions that are each classified as Bach-like (positive-utility progressions). The TB region represents the set of harmonizations that are piece-wise Bach-like (constructed from progressions that are each truly Bach-like). The star region represents the harmonizations that are overall Bach-like. The representative numbers 40 and 4, for in-bounds and positive-utility progressions respectively, are specific to Feature Set 6.

TABLE 2

ESTIMATED PROBABILITY OF GENERATING A BACH-LIKE PHRASE

phrase length (non-cadential chords)	No. of possible phrases			no. of Bach-like phrases	Probability		
	using in-bounds progressions	using positive-utility progressions	using Bach-like progressions		with in-bounds progressions	with positive-utility progressions	with Bach-like progressions
k	40^k	4^k	4^k	3^k	$3^k/40^k$	$0.84^k \times 3^k/4^k$	$3^k/4^k$
1	40	4	4	3	0.08	0.63	0.75
2	2×10^3	16	16	9	0.01	0.40	0.56
3	6×10^4	64	64	27	4×10^{-4}	0.25	0.42
4	3×10^6	3×10^2	3×10^2	81	3×10^{-5}	0.16	0.31
5	1×10^8	1×10^3	1×10^3	2×10^2	2×10^{-6}	0.10	0.23
6	4×10^9	4×10^3	4×10^3	7×10^2	2×10^{-7}	0.06	0.18
7	2×10^{11}	2×10^4	2×10^4	2×10^3	1×10^{-8}	0.04	0.13
8	7×10^{12}	7×10^4	7×10^4	7×10^3	1×10^{-9}	0.02	0.10
9	3×10^{14}	3×10^5	3×10^5	2×10^4	8×10^{-11}	0.02	0.08
10	1×10^{16}	1×10^6	1×10^6	6×10^4	6×10^{-12}	0.01	0.06

current system is not so obvious. Making the score perfect would only increase the probability that a phrase is overall Bach-like by a factor of 2 for short phrases (3 or 4 noncadential chords), and a factor of 6 for longer phrases (8 to 10 noncadential chords). In fact, however, it is very important to decrease the probability of generating a non-Bach-like progression. The reason is that such progressions cannot be screened out in the second (phrase) stage of learning, because that stage will operate in a space of features of a whole phrase (more global features). An 84% Bach-like progression rate means that even for a short phrase (4 noncadential chords) the probability that it will be marred by at least one non-Bach-like progression is 50%. It should be recognized also that the average listener usually will be more disturbed by a bad two-chord progression than by a flaw at a more global level (thus, finishing the job of screening out non-Bach-like progressions is of more basic importance for the quality of the music than implementing the phrase stage of learning).

From the analysis in this section we conclude that the learning done by the clusterer in our current system has made a very substantial contribution towards the goal of being able to generate Bach-like harmonizations of a chorale phrase melody.

E. Other Points of Discussion

1. Effect of increasing the number of training examples

Using Feature Set 2 and Feature Set 3 ChoraLearn was tested first with a set of 155 training examples, and then again when the set was increased to 301. Two things were expected to increase along with the training example size: the number of possible harmonizations for a chorale phrase (using positive-utility chord progressions) and the correctness ("Bach-likeness") of the chord progressions.

In the case of the Feature Set 3 version, the number of possible harmonizations increased substantially. This was not the case with the Feature Set 2 version. In the latter case, the limiting factor was Feature 7 – the trial object had to have a HarmonicProgression that occurred in the training examples in order to have a positive utility. The number of different HarmonicProgressions increased only modestly by doubling the number of training examples. On the other hand, the number of total objects input to the clusterer doubled (because the number of Bach objects doubled, the number of non-Bach objects necessarily doubled also – see Section III. A. 1). Because of this increase in objects, the Bach objects could be concentrated in smaller regions, i.e., the size of the positive-utility regions shrank. As a result, the number of possible harmonizations of the test melody actually decreased (from 11 to 10).

The reason why the positive-utility regions shrank in size with an increase in number of objects has to do with a PLS1 clusterer parameter, mintotal, that specified the minimum total number of objects that a region could contain. This parameter was set at 5 for both the small and large sets of training examples. Thus, for example, in the case of a Bach object that is far away from any other Bach objects, just increasing the average density of non-Bach objects in the space will have the effect of making the region containing the Bach-object and its 4 nearest non-Bach neighbors smaller. The problem of choosing the right value for mintotal will be discussed in Section F, below.

It was easy to see that the smaller positive-utility regions resulting from the increased number of training examples with Feature Set 2 were more correct, because of Feature 8 (the hidden fifths, hidden octaves feature). With the smaller number of training examples (large regions), every region in the space spanned both of the two in-bounds values of this feature. In other words, no region boundaries separated the half of the space in which every object has

a hidden fifth or hidden octave from the other half of the space, even though there was only 1 Bach object with a hidden fifth (and none with a hidden octave). In contrast, with the larger number of examples (smaller regions), there were very few positive-utility regions that allowed the value of Feature 8 corresponding to a hidden fifth or hidden octave.[15]

2. The problem of extraneous features

After running the clusterer with Feature Set 2, I checked to see if three Bach objects that were *not* in the set of training examples mapped into positive-utility regions (this job can be done quickly using a little program called TestEval). One of them did not, although its harmonic progression was the same as that of two objects that were among the training examples. I then determined that it missed being in a positive-utility region because of its value for Feature 2 (bass register). So I simplified Feature 2, distinguishing only the very low bass notes from the rest (2 values), and got rid of Feature 4 altogether. Using this new feature set (Feature Set 4), the clusterer partitioned the space so that the Bach object that evaluated to zero before, now mapped into a positive-utility region. On the other hand, changing Feature Set 3 in the same way (to create Feature Set 5) resulted in the same Bach object being moved from a positive-utility to a zero-utility region. This was determined to have been a case of serendipity.[16]

[15]Among these regions, obviously, were the regions containing the Bach objects with hidden fifths (there were now 2). Many of the other regions that allowed the value of Feature 8 corresponding to a hidden fifth or hidden octave were regions in which it was actually impossible for a hidden fifth or hidden octave to occur because of the ranges of Features 1 and 5.

[16]Removing a feature from a set has two effects which influence the relative amount of space in positive-utility regions in opposite ways. The discrimination between different good objects is decreased, which tends to decrease the relative amount of positive-utility space, and the discrimination between good objects and "bad" objects is also decreased, which tends to increase the relative amount of positive-utility space. With Feature Set 3, there were many regions containing exactly one good object. Because of the minimum total points rule (see Section D above) each of these regions was forced to be large enough to include at least 5 points altogether (adding good and "bad"). Removing Feature 4 and the original Feature 2 resulted in some of the formerly different good points becoming indistinguishable from each other and others becoming more similar to each other. Thus there were fewer positive-utility regions, and some of the regions that contained only 1 good point were replaced by regions containing 2 or more, so that they needed to include fewer "bad" (non-Bach) points. The final outcome was fewer possible harmonizations. (The particular Bach object in question was in a region containing one good training point, using Feature Set 3. In Feature Set 5, that good point became indistinguishable from another good point, and so they obviously were in the same region. Both of these training objects differed from the test object with regard to their values of Feature 9, soprano chord element, and that is why the test object moved from a positive-utility region to a zero-utility region.) In contrast, with Feature Set 2, there were fewer regions containing very small numbers of good points, and therefore the second effect of removing a feature, the de-

Extraneous (irrelevant or weakly relevant) features separate otherwise similar objects. Because of random effects, a system using a feature set that contains extraneous features will in general require more training examples for a given amount of learning than a system using a similar feature set but without the extraneous features.

3. Concept learning with noisy data--how to tune the learning system for best results

If all of the "non-Bach" feature vectors given to the clusterer were non-Bach-like, then it would have been best to partition the feature space in such a way that the Bach and non-Bach vectors would have been completely segregated in separate regions. However, surely some of the non-Bach vectors were in fact Bach-like (see Section III.A.1). Therefore, in order not to make the regions containing Bach vectors (positive-utility regions) overly restrictive, it is a good idea to allow them to contain some non-Bach vectors as well (actually, since there were probably many non-Bach vectors coincident with Bach vectors, it would have been impossible not to allow it).

A PLS1 clusterer parameter "tsubalphaclus" determines how significantly different two parts of a region must be (in terms of the respective numbers of Bach and non-Bach vectors in them—a statistical calculation is performed) in order for them to be separated by a new partition (creating two regions out of the original one). The parameter "mintotal" already mentioned above (Section 1) can also be adjusted ad hoc to deal with noisy data. By putting

creased discrimination between good and "bad" objects, predominated. Thus there was an increase in the number of "bad" points included in positive-utility space, with an outcome of a greater number of possible harmonizations.

It should also be noted that there is a serendipitous effect resulting from any kind of change in the clusterer's input, whether it is a change in the training example file, a change in the mintotal parameter, or an alteration in the feature set. This is due to the fact that the clusterer does not, in general, create optimal clusterings, and the particular outcome, especially when we are concerned with zero utility vs. positive utility, can be substantially different (although approximately equally good) as a result of small changes. The clusterer makes successive optimal splits of regions, starting with the whole space (Rendell, 1983). But each split is permanent. In other words, when the splitting is finished, the clusterer does not go back and try different initial splits to see if the final clustering might be improved (another system, PLS2, which has been used in other domains, optimizes clusterings with respect to performance--see Rendell, 1985). Thus, if, for example, the addition of one good point to the training examples were to result in the first split occurring perpendicular to the feature X dimension instead of to the feature Y dimension, the final clustering might be quite different.

a lower limit on the total number of vectors that a region may contain, a mintotal value prevents a too-small group of Bach-vectors from being isolated. In all the results discussed so far, tsubalphaclus was set at zero and mintotal at 5. In this section I discuss some effects of varying mintotal.

For a given set of training examples and a given feature set, there must be an optimum value or optimal range of values for mintotal. Obviously, increasing the minimum number of objects that a region may have will result in fewer and larger regions. This, in turn, will result in a larger portion of feature space being contained in positive-utility regions. Since we are using utility essentially as a boolean function (Section A above), this means more possible Chords for a given soprano note and NextChord. The effect of increasing mintotal with Feature Set 6 and the set of 301 Bach training examples can be seen in Table 3.

TABLE 3

EFFECTS OF VARYING THE CLUSTERER PARAMETER MINTOTAL

mintotal	cpu time for clustering (hr:min:sec)	no. of regions in feature space	no. of possible harmonizations*
5	1:02:12	295	28
6	57:08	265	80
7	56:58	245	122
10	53:47	204	not measured
20	41:32	129	not measured
30	35:45	97	1406

*of a test phrase requiring 4 chords to be filled in by ChoraLearn (only positive-utility progressions were accepted).

The test phrase used to generate the data in Table 3 was phrase 1 of "O Gott, due frommer Gott" (No. 291 in the Riemenschneider collection). Of the 4 Bach objects corresponding to the 4 chords that were to be filled in by computer, only one had a feature space description identical to one of the Bach objects in the set of training examples (using

either Feature Set 4 or Feature Set 6). The other three did, however, have HarmonicProgressions that occurred in the training examples. With Feature Set 4, it was possible to find a value for mintotal that was large enough so that all three of those objects mapped into positive-utility regions, but was also small enough that some regions were separated by boundaries along the dimension of Feature 8 (an indication of discrimination – see Section D above). This was not possible with Feature Set 6.

A good way to choose a value of mintotal might be to take a larger set of Bach test objects (that were not in the set of training examples), say 10 or 20, and a similar-sized set of carefully chosen "bad" objects (prepared by someone very familiar with Bach's style). A value for mintotal would then be chosen that resulted in the greatest number of correct classifications (Bach objects should get a positive utility and bad objects a zero utility). If one were more concerned about not generating bad progressions than about finding all the good ones, one could, for example, increase the relative importance of correctly classifying the bad objects with some weighting factor. Such a calibration of the clusterings was not performed in the present study. As we will see in Section V below, the clusterings of the Bach data that have been discussed here should be regarded as a "rough sketch" of the concept "Bach-like progression." No matter how we choose the clustering parameters, the result will need refinement. In Section V a way of achieving such a refinement will be proposed.

4. Sources of bias

As Watanabe (1969) has proven, and Mitchell (1980) reiterated, bias, which influences what generalizations will be made from a set of examples, is essential (unavoidable) in making any generalization. In this small section, I wish to point out just two of the sources of bias in ChoraLearn.

The choice of features is an obvious source of bias. The choices of features I made were based on my limited knowledge of music and my own biases as to what was important. In addition, I made a decision that the system should be quite general in the melodies it could harmonize. Therefore, I avoided using soprano scale element as a feature, for then if a melody contained scale elements that were rare or absent from the training examples, it would be difficult, or, impossible for the system to harmonize it. Similarly, the interval between consecutive soprano notes (Feature 5) was given only 7 possible values, with intervals absent from the training examples being grouped together with rare intervals.[17]

Another source of bias is due to the definition of the problem (Section II.B). The problem was chosen as a piece of the larger problem of generating a complete chorale from a given melody. This larger problem might have been broken up quite differently, if, for example, one were to have used Schenkerian analysis as in Ebcioglu's (1984, 1986) expert system.

[17]The same goal, generality, might have been achieved by interspersing values for absent soprano intervals (or scale elements) with those that occurred in the training examples. Because of the way non-Bach examples were generated, there would have been no objects at all (neither Bach nor non-Bach) having values absent from the training examples. Since the clusterer does not allow regions without any objects, regions containing objects with neighboring values would have been extended to include the absent values.

V. PROPOSED USE OF FEEDBACK FROM AN EXPERT USER

A. The Problem

In the preceding sections I have explained that in ChoraLearn's use of the PLS clusterer, any hyperrectangle (region) in the clustered feature space that has a positive utility is considered essentially good (see especially Section IV.A). We would like, ideally, to have a perfectly partitioned feature space, i.e., one in which the feature vector for every good object is located in a good region, and the feature vector for every bad object is located in a bad (zero–utility) region. The importance of being able to come even closer to this goal was emphasized in Section IV.D.

It is to be expected that increasing the number of training examples given to the clusterer would help. However, imperfections in the features may demand a lot of data. (The features used by ChoraLearn are imperfect in that they are mostly rather "primitive," i.e., they are for the most part basic features rather than appropriately chosen conglomerates of basic features,[18] and they have "rough" spots, i.e., adjacent feature values along a dimension do not always represent a high degree of similarity.) And even nearly ideal features may demand more data than are easily available. For example, a human Bach expert harmonizing a chorale in the style of Bach draws on his knowledge not only of Bach's chorale harmonizations, but of Bach's other works, of the works of Bach's contemporaries and predecessors, and of rules that were established by the time of Bach and were part of the education of the composer. If Bach had harmonized only one chorale, it would still be possible to talk about harmonizing chorales in the style of Bach, but the data contained in the one chorale harmonization would clearly be inadequate.

---

[18] Being primitive, the features are also highly interdependent, i.e., the "goodness" of a particular feature value depends upon the values of the other features. (My use of the term, interdependent, is perhaps misleading. Two features could be independent random variables and yet be interdependent in the sense I mean.)

B. The Solution

It seemed that a human Bach expert, if she were to use the ChoraLearn system, could provide some feedback to the system that could be used to improve subsequent performances. We have arrived at a simple way to use this feedback, and our method takes advantage of the way the clusterer works. It is explained in the following paragraphs.

The presence of even a single Bach progression ("good point") in a region makes the region acceptable. One thing that would be useful to have is a set of equally powerful "bad points" that would carve out "bad regions" in the same way.[19] For practical purposes, we only need to eliminate bad subregions that lie within the current good regions. This can be achieved as follows. (I will explain the method as it would work with ChoraLearn, but its generality will be obvious.)

The expert user (human Bach expert) would give ChoraLearn a melodic phrase to harmonize. She would then examine a set of output harmonizations (if the phrase were small, this might be the set of all possible harmonizations using acceptable progressions; otherwise, it would be a set generated by the "OKChord" or "GoodChord" choice procedures). When she found a progression she thought would be bad (un-Bach-like) in every conceivable context, then the feature vector for that progression would be added to a list of bad progressions.

The proposed feedback-handling system would do two things with a vector on the "bad list." First, it would check if the vector were also on the Bach list (from training examples). If so, then it would remove it from the Bach list, perhaps first prompting the expert user to think again about the quality of progressions having the feature vector in question. In this

[19] The non-Bach progressions in the input file for the clusterer cannot serve as the desired bad points. They clearly contain some good points, since Bach could have written, and sometimes did write, several different harmonizations for a chorale phrase.

way, mistakes that were in the training example file could be corrected.[20]

The other thing that the system would do with objects on the bad list is use them for a separate clustering against the objects on the Bach list. In other words, in a fresh copy of the feature space, the good objects (Bach list) and the newly discovered bad objects (but not the non–Bach objects from the original feature space) would be mapped and then clustered. Because of the checking procedure described in the previous paragraph, no bad object could map into the same point as a good object. Therefore, if we set the clustering parameter, mintotal, equal to 1, then each of the resultant regions will contain either good points or bad points, never a mixture.[21]

In subsequent performances (uses of ChoraLearn to harmonize a melody), the new feature space would be used as follows. If a trial object maps into a positive–utility region in the first feature space (which may have been changed due to removal of bad objects from the Bach list and reclustering), then it is mapped into the new feature space. If there it maps into a region containing bad points, it is unacceptable (reevaluated to zero utility); otherwise, it is acceptable. Use of this new feature space as a final screen in this way has the nice characteristic of being partly deterministic and partly probabilistic. It is deterministic in that no object having the same feature vector as the object that was deemed bad by the expert user can ever again be acceptable. It is probabilistic in that some other feature vectors that are clustered together with discovered bad points also become unacceptable.

The user can also easily add new, supplementary, objects to the Bach list. If the melodic phrase she gives to ChoraLearn to harmonize is a phrase that Bach harmonized (but

---

[20] This procedure depends on the feature set being complete enough to always distinguish a bad object from a good one.

[21] In truth, certain configurations could result in a mixed region, but the likelihood of such a configuration occurring is remote. If such an instance were to occur, the system could decide to classify the region as bad and sacrifice the ability to use the good points.

is not in the original training examples), then she can ask the system to check if the vector of each progression of the Bach version is in the current Bach list. If not, she can simply add the phrase to the Bach example file.[22]

It is hoped that going through this cycle of testing and correcting a few times will result in a greatly improved system that is capable of generating most of the progressions Bach would have used, and only very rarely generates a bad progression. A nice aspect of the proposed feedback mechanism is that an expert user can improve the program without doing any programming.

One might think that it would be most efficient to deal only with Bach points and bad points, and to omit the clusterer's action on Bach and non–Bach points. This is probably not the case. The initial running of the clusterer achieves a great deal, and does so rather rapidly. With Feature Set 6 and an input file containing 301 Bach training progressions, 10998 of the 12103 total progressions (Bach plus non–Bach), or 91%, were clustered into regions of zero utility ("bad" regions). This is an accurate estimate of the percentage of trial progressions that will be eliminated by the clusterer's action. In other words, the expert user needs to identify only a very restricted set of bad objects: those that map into the small portion of the feature space that the clusterer thinks is good. Thus, in the new system we are proposing, learning from a set of training examples makes a rapid sketch of the concept, and learning from feedback given by an expert user fills in the details with a fine brush.

[22] If we are concerned that the utilities of the regions reflect the frequency with which Bach used objects in them, then it is better to add a whole phrase, with associated non–Bach objects, than just a single progression.

VI. WHERE DO WE GO FROM HERE? A BRIEF SUMMARY OF THE TASKS THAT REMAIN FOR A COMPLETE CHORALE-WRITING PROGRAM

We have approached the task of making a chorale out of a given melody in a way similar to that of Thomas in her rule-based system, Vivace (Thomas,1985); i.e., by deciding that it should be broken up into modular subtasks. The subtask of writing a figured bass for the noncadential portion of a phrase we have divided further into the serial operations of writing a simplified figured bass (quarter notes only) and then adding eighth notes where appropriate. The first of these, writing a simplified figured bass for a phrase, we decided to attack by first generating a set of simplified figured basses that are piecewise good (from one note to the next) and then from these selecting those that are overall good. We have come close to succeeding in producing a set of piecewise good simplified figured bass harmonizations for the noncadential portion of phrases in major keys.

At this point, we could also very easily extend the performance to minor keys. All that is needed is a MinorKeyScreen (analogous to the MajorKeyScreen described in Section III.A.1) and a set of training examples in minor keys.

The proposed feedback-from-an-expert-user mechanism described in the last section is expected to remove the errors in the piecewise good harmonizations. When this is completed, there remain some decisions to be made as to how to proceed in finding overall-Bach-like harmonizations of phrases. For example, in the case of long phrases, it might be efficient to first select measures (sequences of three or four chords) that are overall Bach-like and use these to build sets of phrases that are measure-wise Bach-like, from which overall-Bach-like phrases would be subsequently selected. An alternative would be to have a wider "sliding window" (the current system has a "window" that is two chords wide, because it examines

features of two-chord progressions, but such a window could be widened to examine three or four chords of a developing phrase). This sliding window alternative would involve more second-level evaluations than the measure-by-measure approach.[23]

The other major subtasks, besides writing figured basses for noncadential portions of phrases, are filling in the middle voices (alto and tenor) and writing the cadences. Then there is the problem of joining the phrases together. This might be achieved simply as follows: consider the soprano note of the final cadence chord of a phrase (phrase A) as the first note of the next phrase (phrase B). Then accept only harmonizations of phrase B that have as their first chord the same chord as the already determined final cadence chord of phrase A.

Finally, there is the most global task of deciding what key to harmonize each phrase in, and where to make modulations (changes of key).

Probably none of the remaining pieces of the chorale-writing task is more difficult than the one that ChoraLearn now can almost do correctly, after being exposed to only a small number of training examples. The possibility of being able to develop a complete chorale-writing system based on learning with the PLS1 clusterer is therefore likely.

---

[23]I am assuming that two feature spaces would be involved: the current one, which uses a two-chord-wide window, and a new one that would use a wider window. One might imagine that we could instead use just one large feature space that had all the features of the current feature space plus additional features to extend the descriptions to longer sequences of chords. I believe, however, that this would drastically slow down learning, i.e., the number of training examples would have to increase greatly. It is important in attacking large problems like chorale harmonization to keep each of the modular subtasks to a manageable size (divide and conquer).

VII. CONCLUSIONS

Ebcioglu, in his 1986 thesis on his rule-based harmonization system, stated that it is unreasonable to expect that an artificial system could be developed that would be capable of "writing its own rules." I think that what we have seen with ChoraLearn indicates that a machine learning system *can* achieve something in the problem domain of harmonization. Ebcioglu's argument is that the task is too large. But like most large tasks, the task of chorale writing can be broken down into smaller, manageable tasks. We have started with a piece of the task that is manageable for the PLS1 clusterer, but which is nontrivial (see Section I.D). Of course, the refinement method we suggested in Section V, using feedback from an expert user, means that a live body will be used to help ChoraLearn "write its own rules." Still, there is an important conceptual and also practical distinction to be made between what we are proposing and the use of rule-based methods. What we are proposing is a way that an expert user can improve the system without any new code having to be written.

Because a learning approach to chorale writing (or to any problem) makes inherently fewer demands for programming than a rule-based approach, it should prove to be cheaper and faster to develop. There are two additional advantages of our approach over a rule-based one. First of all, once the system is developed it has great flexibility; a different set of training examples (e.g., from a different composer, or from a specific period of Bach's life) will rapidly result in a system that harmonizes in a different style. Second, the feature space representation of rules used by the clusterer might be able to represent some "rules" of Bach's (or another artist's) style that would be quite complex and, therefore, difficult for a music theorist or programmer to discover and formulate in words. As D. Michie has been quoted as saying (Lamb, 1984), "Really expert knowledge is intuitive, and it is not necessarily

accessible to the expert himself." Thus, it is possible that the learning by clustering approach will ultimately yield a performance that is truer to the master.

REFERENCES

Buchanan, B. G., Johnson, C. R., Mitchell, T. M. and Smith, R. G. (1978) Models of learning systems, in Belzer, J. (Ed.), *Encyclopedia of Computer Science and Technology, Vol. 11.* New York: Dekker, 24–51.

Ebcioglu, K. (1984) An expert system for Shenkerian synthesis of chorales in the style of J. S. Bach. *Proceedings of the International Computer Music Conference, 1984,* 135–142.

Ebcioglu, K. (1986) An expert system for harmonization of chorales in the style of J. S. Bach. Doctoral Dissertation, State University of New York at Buffalo, Buffalo, New York.

Gross, D. (1984) Computer applications to music theory: a retrospective. *Computer Music J. 8,* (4) 35–42.

Hiller, L. (1970) Music composed with computers -- a historical survey, in: Lincoln, H. B. (Ed.), *The Computer and Music.* Ithaca, New York: Cornell University Press, 42–96.

Hiller, L. (1981) Composing with computers: a progress report. *Computer Music J. 5,* (4) 7–21.

Hiller, L. A. Jr. and Isaacson, L. M. (1959) *Experimental Music Composition with an Electronic Computer.* New York: McGraw Hill.

Lamb, J. (1984) In pursuit of AI. *Datamation* (May 1) 139–140.

Michalski, R. S. and Stepp, R. E. (1983) Learning from observation: conceptual clustering, in: Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto, California: Tioga, 331–363.

Mitchell, T. M. (1980) The need for biases in learning generalizations. Technical Report CBM–TR–117, Department of Computer Science, Rutgers University, New Brunswick, New Jersey.

Piston, W. (1962) *Harmony, Third Edition.* New York: Norton.

Rendell, L. (1983) A new basis for state-space learning systems and a successful implementation. *Artificial Intelligence 20,* 369–392.

Rendell, L.A. (1985) Genetic plans and the probabilistic learning system: synthesis and results. *Proceedings of an International Conference on Genetic Algorithms and their Applications* (Carnegie-Mellon University), 60–73.

Rendell, L.A. (1986a) A general framework for induction and a study of selective induction, *Machine Learning Journal 1,* in press.

Rendell, L. (1986b) Conceptual knowledge acquisition in search, in: Bolc, L. (Ed.), *Knowledge Based Learning Systems.* Berlin: Springer–Verlag, in press.

Rendell, L.A. (1986c) Induction, of and by probability, in Kanal, L.N. and Lemmer, J. (Eds.), *Uncertainty in Artificial Intelligence.* Amsterdam: Elsevier North–Holland, 429–443.

Rich, E. (1983) *Artificial Intelligence.* New York: McGraw–Hill.

Riemenschneider, A. (Ed.) (1941) *371 Bach Chorales and 69 Chorale Melodies with Figured Bass by Johann Sebastian Bach.* New York: Schirmer.

Roads, C. (1980) Artificial intelligence and music. *Computer Music J. 4,* (2) 13–25.

Rothgeb, J. (1980) Simulating musical skills by digital computer. *Computer Music J. 4,* (2) 36–40.

Segre, A. M. (1981) A system for generating four–voice chorale style counterpoint using artificial intelligence techniques. *Proceedings of the Quarto Colloquio di Informatica Musicale, 1981.*

Terry, C. S. (Ed.) (1929) *The Four-Part Chorales of J. S. Bach.* London: Oxford University Press.

Thomas, M. T. (1985) Vivace: a rule-based AI system for composition. *Proceedings of the International Computer Music Conference, 1985,* 267–274.

Watanabe, S. (1969) *Knowing and Guessing: A Quantitative Study of Inference and Information.* New York: Wiley.